2013

A Federated Architecture for Managing Health Information in Ethiopia

Russell Gillen

A Federated Architecture for Managing Health Information in Ethiopia

A Thesis Submitted in Partial Fulfillment of the Requirements

For the Degree of Master of Science in Computer Science

University of Southern Maine

Department of Computer Science

By

Russell Gillen

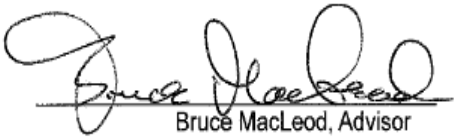
2013

The University of Southern Maine

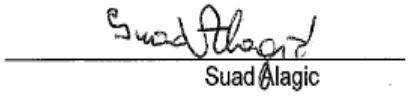
Department of Computer Science

January 2014

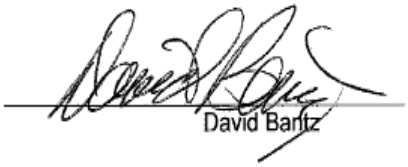
We hereby recommend that the thesis of Russell Gillen entitled A *Federated Architecture for Managing Health Information in Ethiopia* be accepted in partial fulfillment of the requirements for the Degree of Master of Computer Science



Bruce MacLeod, Advisor

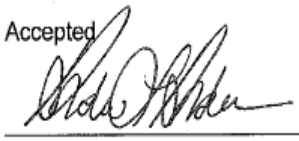


Suad Alagic



David Bantz

Accepted



Dean

College of Science, Technology and Health

ACKNOWLEDGEMENTS

First I'd like to express my thanks to Professor Bruce MacLeod for being a great mentor and colleague to work with. I have learned a tremendous amount about software development (and writing) during my time with Bruce and those experiences have been invaluable. I'd also like to thank all of the other professors and teachers that have aided me along the way and helped to stimulate me intellectually. And I am thankful to all of my friends and family for encouraging me and asking about my work when they didn't have any idea what most of it meant. Especially my mother's allusions to “the platform”! The University of Southern Maine has granted me the opportunity to work on a world class project such as MOTECH and I am forever grateful for that. Many of the lessons I have learned during my time at the University will be with me for the rest of my career. Thanks everyone!

ABSTRACT

Electronic health systems can be leveraged to aid in the delivery and management of health care for patients in developing countries. The delivery of health information by mobile devices is playing an increasing role in health information management. Mobile health software systems can be utilized to target under-served populations for health care administration by providing messaging and alerting for reminders, informational messages, patient compliance and more. MOTECH is a software framework developed primarily for aiding health care delivery in under-served populations. This thesis utilizes and extends the MOTECH software framework to meet the requirements for a health systems pilot project in Ethiopia. The pilot project also serves as a model to help drive the development of the MOTECH platform.

Table of Contents

List of Figures

I. Introduction

1.1 - Electronic Health Systems

Health systems help providers, institutions, and governments facilitate the delivery of health care to patients. Prior to the development of electronic medical records, health information systems were paper based. Paper based health systems often face many challenges, including error correction, organizational and logistics chains, sharing data between institutions or even within a single facility, and difficulty in reporting and management. Electronic health systems address many of these challenges and offer new features otherwise not available with paper based systems [1].

Electronic health record systems help improve coordination, accuracy of diagnosis and treatment, medical compliance, patient participation, quality, education, convenience and cost savings. Decision based care can be automated through software systems when it would otherwise require manual intervention by a provider. Messaging and alerting, analytical reports and much more can be achieved by adopting an electronic based health system. A number of paper based health systems around the world, including those found within developing countries, are transitioning to electronic health systems [2]. Re-usable and extensible software platforms can help alleviate the need to build an electronic health information system from scratch and one such project, MOTECH, aims to achieve this by supporting numerous use cases that

are primarily related to health care delivery in low resource settings.

1.2 – Health care delivery using mobile phones in developing countries

Health care delivery in developing countries poses many challenges. With the rapid increase in mobile phone usage, numerous strategies are being explored to leverage this technology. Pilot projects are using mobile devices as information systems for health workers, interactive voice response (IVR) phone calls are being placed directly to patients, and SMS based messaging is helping improve operational logistics for drug supplies [4, 5, 6]. The use of Mobile devices continues to grow world wide and software applications represent a burgeoning subsection of the computing industry. It is this proliferation and increasing access for lower income regions that has made mobile handsets a focal point for improving health care delivery.

There is a large organizational disparity between health care systems found in developed countries and in developing countries. In developed countries, regulation and laws have established a complex and intricate ecosystem that new software systems must navigate. Attempts to standardize and streamline electronic health systems have been underway for years, which have met some success. The HLA Clinical Document Architecture (CDA) has multiple implementations attempting to address this issue by providing an XML standard for communicating health information [3]. Electronic health information standards, regulations and a greater adherence to legal requirements can require significant

investment for successfully deploying mobile health applications in developed countries. Developing countries are struggling to determine which standards to adopt, if any, and many of the projects are smaller and do not adhere to any regulatory standard. In some respects, software development of health information systems in developing countries can be easier because there are fewer legal and standards-based hurdles to cross during implementation [4].

Short Message Service (SMS) text messages are playing an increasingly larger role in mobile health care delivery around the world. Sending and gathering medical information, expert responses for user questions, patient to provider connections and transactions are just some of the ways text messaging on mobile phones are being used in the health and medical field. One of the most common use cases for medical text messaging is for notification and education outreach, including educational messages on topics from pregnancy or health tips, as well as reminder notifications for pills, appointments or even a reminder to drink water [5].

In 2011 a mobile health summit was held in Africa to assess the current state of mobile health. While many participants agreed on the useful health care delivery mobile devices can provide, one government minister from South Africa called for caution over potential pitfalls, including regulation, confidentiality and cost to patients. However, the benefits of mobile health are already being realized. The World Health Organization (WHO) released a report showing that 83% out of the 122

surveyed countries use mobile phones for free emergency calls, text messaging for pill reminders, dissemination of health information and transmission of test and lab results. The WHO has set up the Global Observatory for eHealth in Geneva and reports that up to 40 African countries are already using mobile health services [6].

Mobile phones are not only being used to inform both patients and physicians, they are also being used by nurses and less experienced health workers that sometimes are in need of reminders or informational messages themselves. One study has examined how to improve health worker performance through automated SMS. Community health workers typically receive limited training in their role as an interface between the medical health system and the community. These workers are usually from the community they are serving and sometimes contribute as volunteers. The study used an escalating reminder system that sends SMS text message reminders to the health workers until finally notifying his or her supervisor after several late days. These reminders resulted in an 86% reduction in the average number of days that the clinical workers were late with their reports or submissions [7].

Veterans Affairs Ann Arbor Healthcare System and the University of Michigan teamed up to conduct a study on cell phones and their effects on health care in less developed countries. The study was conducted in a low income region of Honduras and concluded that mobile phones could help low-income patients manage diabetes and other chronic diseases. Diabetics were enrolled into the program using low-cost Internet-based

cell phones that interacted with a back-end cloud service. Individual patients received automated, interactive phone calls on a weekly basis. The researchers discovered that these phone calls provided significant improvement to patient diabetes management and general health. A documented, clinically important improvement was observed in patient blood sugar control. Of those participating in the diabetes management service, 92% expressed that they would use the service again. The effects of mobile health are already underway and successful pilots in traditionally less tech savvy regions point to a major role of mobile devices in future health care outcomes [8].

Mobile health is not progressing without facing challenges. A study in Pretoria, South Africa examined the adoption and sustainability of mobile health phones by practitioners and patients. The researchers discovered that the success of a program is largely dependent on patient and care giver willingness to adopt and learn new technologies, the cost of these technologies (such as phone service expenditures), and opinions regarding government sponsored services. One of the major challenges of mobile health going forward is the human element. The study determined that many health workers are reluctant to abandon their paper based systems in favor of a new, untested paradigm shift [9].

Patty Mechael, executive director of the U.N. Foundation's mHealth alliance, as well as others, claim that organizations must attempt to filter out the feasible, useful applications of mobile health from those that are not. There are many challenges in developing national mobile health

systems, including lack of reliable energy sources to power the phone chargers in low income countries, scalable information systems, and understanding the efficacy of pilot interventions. While there are numerous pilot programs throughout the world, Mechael argues that we need to be “more strategic, collaborative, cohesive” in our approach to mobile health. Josh Nesbit, CEO of Medic Mobile, points out the large media coverage of mobile health in recent years and how it may appear that millions of health workers have already integrated mobile phones into regular practice, while the reality of the situation is that the number is in the tens of thousands [10].

1.3 - MOTECH

MOTECH is an extensible open-source software platform originally developed by the University of Southern Maine and being led by the Grameen Foundation that primarily targets electronic health care delivery and messaging. MOTECH is a modular based software system and is the foundation for the work completed during the course of this thesis. MOTECH provides many features, including IVR and SMS messaging, alerting, electronic medical record support and customized schedules of care for patients. MOTECH is a generalized framework that is intended for use by implementations that are able to leverage the platform's features to accomplish project specific requirements.

1.4 - Ethiopia Pilot Project

As part of the work completed during this thesis, a health information software pilot project for Ethiopia was undertaken in cooperation with Andy Kanter from Columbia University. The project focused on two areas of health information: improving health indicator reporting for regional facilities across Ethiopia and collecting individual level health data for pregnant mothers and their children by utilizing mobile phones.

During the course of the thesis a software implementation of MOTECH was developed and successfully deployed in Ethiopia. The project was deployed in two phases: an indicator reporting phase that leveraged MOTECH for health facility compliance and messaging, followed by a proof of concept demonstration presented by Andy Kanter of Columbia University for the Ethiopian Ministry of Health. During the work of the thesis we identified three key challenges throughout the process: interoperability between software systems, mapping data between two disparate data models, and consolidating the identity of health providers and facilities across software systems. We addressed these challenges by developing various software modules within the MOTECH platform to provide the necessary functionality to meet the project's requirements. The details of these modules are described in the implementation and methodology section.

II. Background

2.1 - MOTECH

Mobile Technology for Community Health (MOTECH) began as a project to aid the quality and quantity of health care for pregnant mothers and infants in Ghana. The initial pilot study also seeks to determine whether health information delivered by mobile phones can improve the health outcomes and quality of care in rural clinics. The project was a partnership between Ghana Health Service, Grameen Foundation, Columbia University and a group at the University of Southern Maine. Launched in 2010, this first iteration of MOTECH has two main applications: a Mobile Midwife application and an application for nurses. The Mobile Midwife application's main function is to send SMS or voice messages to pregnant women and their families with time sensitive information related to pregnancy on a week by week basis. These informational messages are provided in multiple languages. Included among the messages are alerts and reminders for health treatments, actionable information and educational information.

The nurse's application records and tracks the care delivery for women and newborns in a community health worker's area. A nurse can enter data into a mobile phone regarding a patient's clinic visit which MOTECH will use to make health care related decisions for sending health service reminder messages. Health workers are alerted of overdue patient treatments which can result in real world follow ups by the health

worker. Additionally, MOTECH can generate monthly reports for district and regional management offices. This first MOTECH incarnation did not meet scalability requirements that a more flexible, modular architecture could provide. The ideas behind the original MOTECH system have driven a more generalized platform that can be utilized in a wide range of health care projects and potentially used in other business domains such as agricultural management [11, 12, 33].

2.2 - Second Generation MOTECH

A new effort to develop a generalized, multipurpose platform is underway by the Grameen foundation and other partners, including the University of Southern Maine. The new MOTECH platform is a modular software architecture that allows implementers to choose the features they need and provides a path for making use of project specific business logic. By providing a template for implementers to extend, the new MOTECH platform reduces the need to architect and build a mobile health system from the ground up.

As part of the work on this thesis, we have extended the second generation MOTECH architecture to support the use case requirements of a pilot project in Ethiopia. New requirements that arose during the work of the thesis have provided feedback that has helped drive development and design of the platform. We generalized many of these requirements and added the code back into the platform, therefore allowing other implementations to meet these same use case requirements.

MOTECH's Architecture

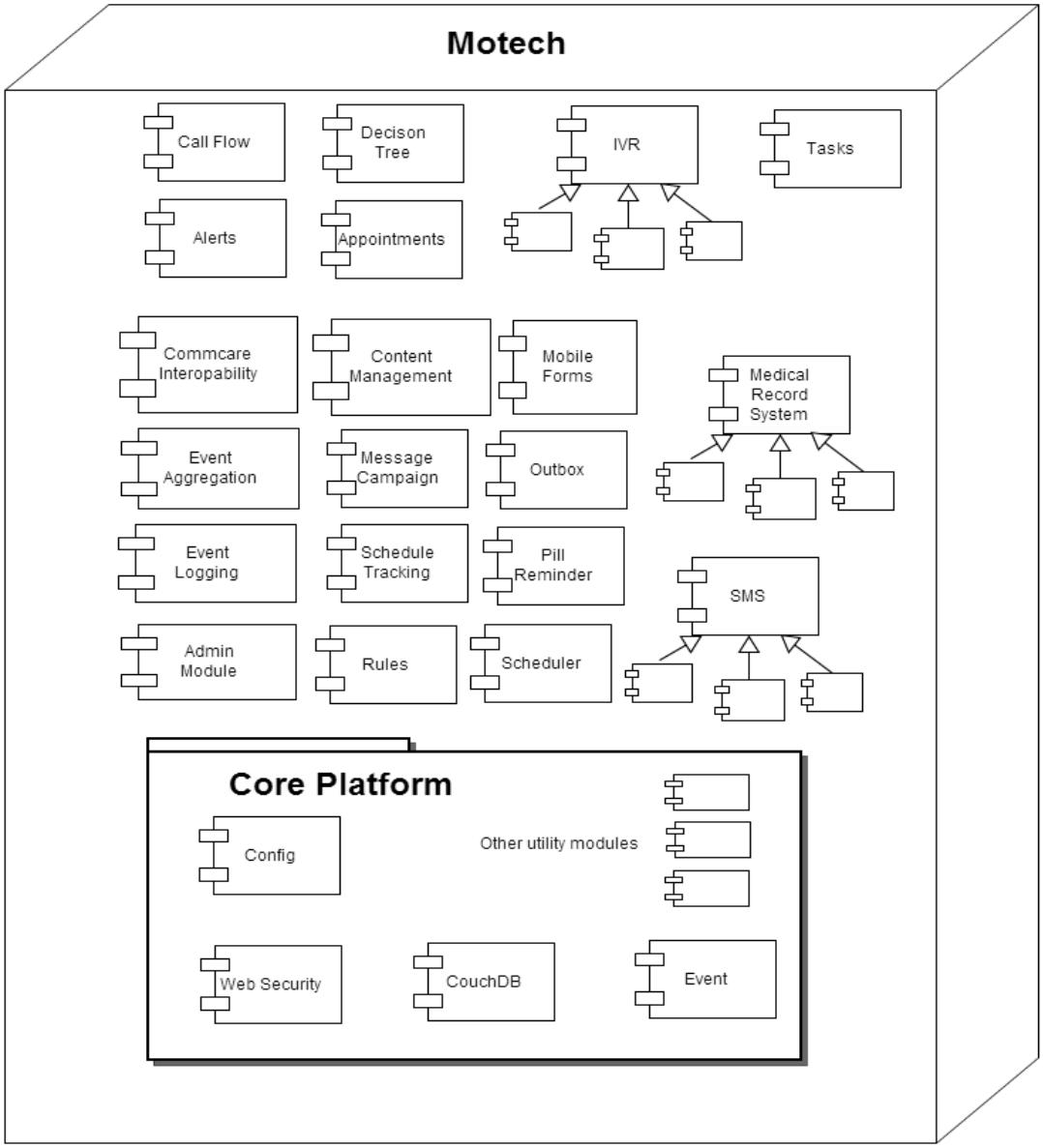


Figure 2.1: MOTECH's modular architecture

2.2.1 - MOTECH as a modular architecture

The MOTECH application is a modular architecture that resides in an Open Services Gateway initiative (OSGi) framework. OSGi is a specification to enable modular deployment of Java applications. A modular architecture separates functionality into self-contained

components that can be maintained and developed with low coupling between separate components residing in the system. MOTECH consists of a core platform, which represents the minimum amount of infrastructure required to run the system, and also includes additional platform modules that can be included and configured to achieve the desired functionality for an implementation. Other custom modules can also be installed within the OSGi framework as long as they conform to OSGi standards [13].

MOTECH uses the OSGi specification to stop, start and update individual modules during deployment. Leveraging the OSGi specification allows the system to stay up while pieces of it are incrementally updated or enhanced. OSGi also manages different versions of software packages within the same run time environment, allowing for the installation of modules that depend on different versions of the same library to co-exist without conflicts. The aim of OSGi and modular development in general is to reduce development time and maintenance costs [13].

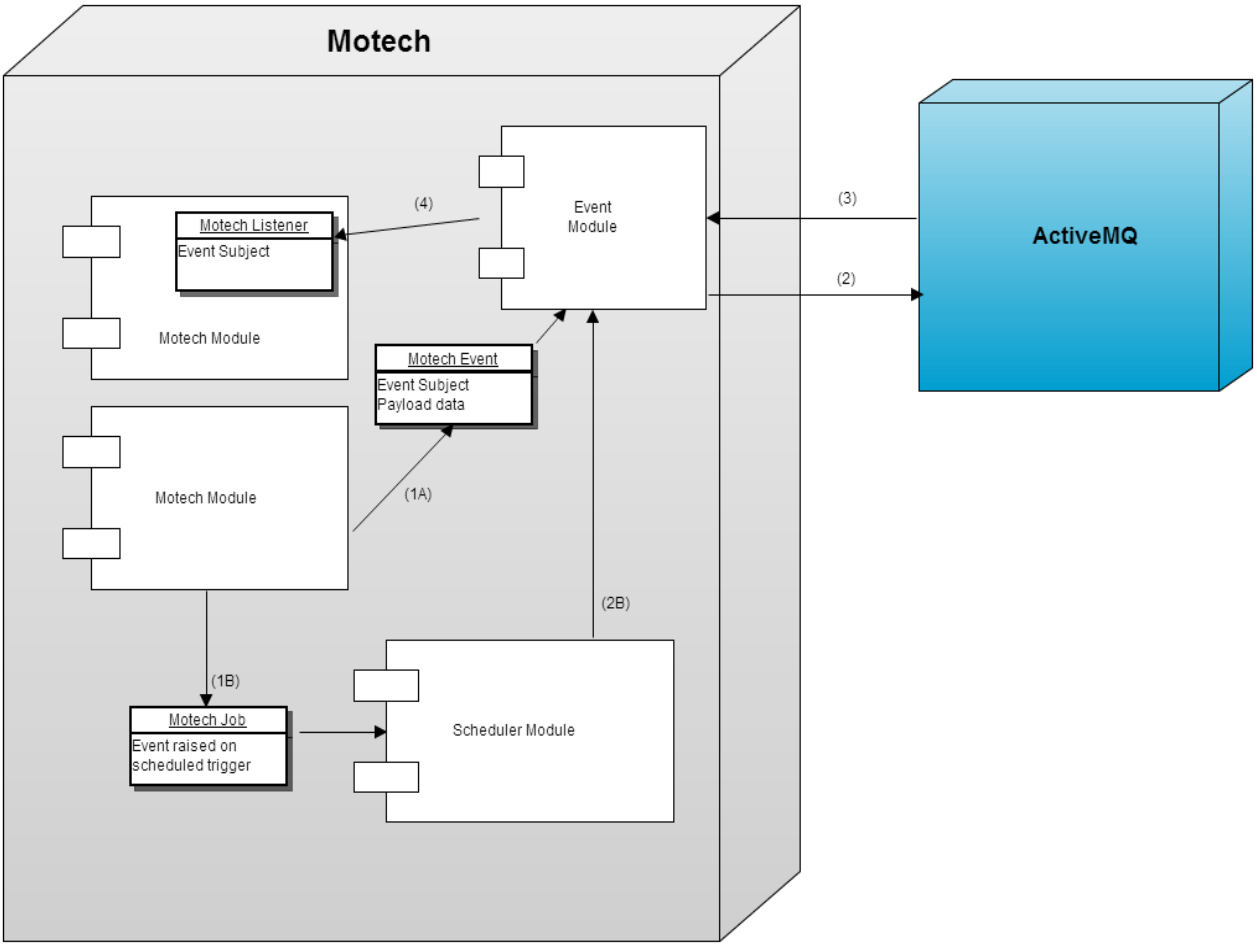
Figure 2.1 demonstrates MOTECH's modular architecture. MOTECH is split into a number of OSGi bundles residing within an OSGi container. These bundles are further split into core platform bundles and optional implementation bundles. The core platform bundles are necessary for the event architecture, security, configuration and other critical infrastructure components needed for the system to operate. Implementation bundles provide optional features that implementers can utilize to enhance their project, such as IVR and SMS capability, an electronic medical record system and customized schedules of care for

patients.

Many of MOTECH's modules publish services through an OSGi service registry, conforming to the service locator pattern [14]. Other modules do not have to be aware of the implementation details of the service to utilize it. For example, there are three modules that provide the same medical record system features: one module that interacts directly with the OpenMRS SQL database, another that communicates with OpenMRS through its web services module, and another that acts as a basic medical information repository internal to MOTECH using CouchDB. Swapping between modules should be seamless because other bundles within the framework consuming their services are unaware of the underlying implementation details.

2.2.2 - MOTECH's event driven messaging system

MOTECH is an event-driven software system. Event-driven systems make use of the observer pattern for event publishing and subscribing [15]. Events occur in isolation and represent key, actionable information taking place in MOTECH, such as a late status for a schedule of care, a new CommCare form's arrival, a medical encounter's creation or the end of an IVR call. Many events are raised by the platform and it is expected that implementations will decide what actions, if any, should be taken based upon these events.



1A. A MOTECH module raises an internal event through an event relay service
1B. Alternatively, a MOTECH module schedules a job that will then later raise a MOTECH event when the job is triggered
2B. The scheduler will raise an internal event in the same fashion a MOTECH module does
2. The event is sent through a JMS channel to ActiveMQ
3. Events are passed back to MOTECH and can be re-raised (2) if there are multiple listeners for the same event, this time spltiting each individually for a particular destination
4. Each event is individually dispatched to a registered MOTECH listener

Figure 2.2: MOTECH's Event Driven Architecture

MOTECH uses its event relay to publish events along with any relevant data as part of their payload. Each event must define a subject which represents its identity. Interested modules or pieces of the system

register listeners on an event subject or set of event subjects. When an event is raised within the system it is dispatched to all registered listeners one by one. The event relay shuttles the events to a JMS server (ActiveMQ) which forwards them back to MOTECH for processing. Figure 2.2 is an architectural diagram detailing these features of the event system within MOTECH.

There are project specific requirements that sometimes can not be met with the features provided by the MOTECH platform. When a project needs to add functionality to the system, code must be written in a custom module to extend and interact with the platform's events and listeners. Figure 2.3 demonstrates how the process is achieved in custom code:

- **Raising an event**

```
if (formStub != null) {
    MotechEvent formEvent = new MotechEvent(EventSubjects.FORM_STUB_EVENT);

    formEvent.getParameters().put(EventDataKeys.RECEIVED_ON, formStub.getReceivedOn());
    formEvent.getParameters().put(EventDataKeys.FORM_ID, formStub.getFormId());
    formEvent.getParameters().put(EventDataKeys.CASE_IDS, formStub.getCaseIds());

    eventRelay.sendEventMessage(formEvent);
} else {
    logger.error("Unable to parse form stub: " + body);
}
```

Registering a custom listener

```
@MotechListener(subjects = EventSubjects.FORM_STUB_EVENT)
public void handleStubForm(MotechEvent event) {

    Map<String, Object> parameters = event.getParameters();

    String formId = (String) parameters.get(EventDataKeys.FORM_ID);

    CommcareForm form = null;

    if (formId != null && formId.trim().length() > 0) {
        form = formService.retrieveForm(formId);
    }

    ...
```

Figure 2.3: MOTECH code example of events and listeners

2.2.3 - MOTECH suite as a federated architecture

MOTECH resides within a federated architecture to interoperate with other external systems. This grouping of various systems has been referred to as the “MOTECH suite”, where MOTECH acts as the key middle software component that helps facilitate the transfer of data between each system in addition to its various other roles as a messaging and scheduling framework. MOTECH can communicate with a variety of systems, including CommCareHQ, OpenMRS, an array of SMS and IVR providers, ActiveMQ, and CouchDB. Figure 2.4 illustrates MOTECH's federated architecture and how a health worker and user might interface with the system. A MOTECH deployment is not limited to the above features and may also choose to exclude interoperability modules if it does not make use of these external systems.

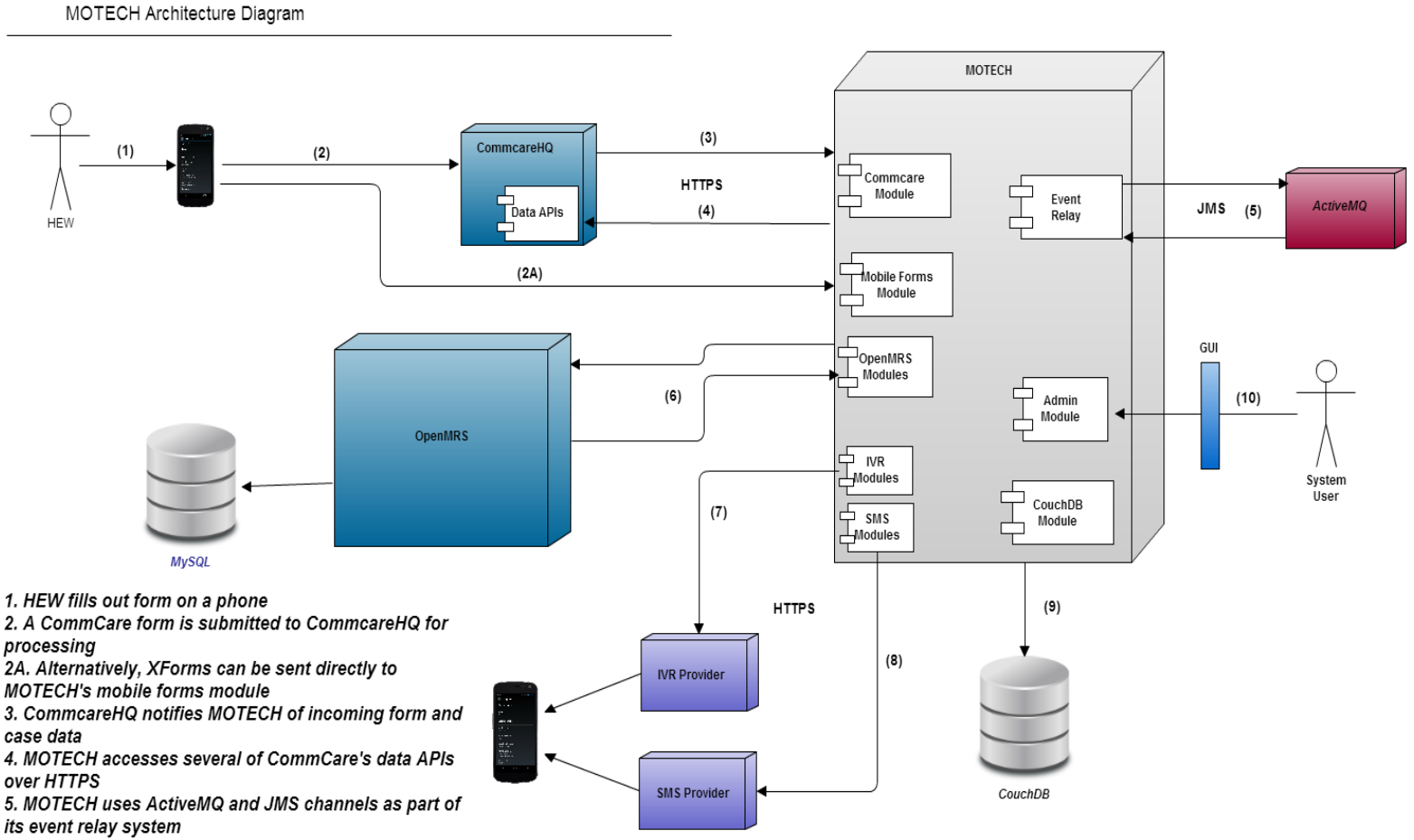


Figure 2.4: MOTECH's Federated Architecture

2.3 - CommcareHQ

2.3.1 - Overview of functionality

CommCareHQ is an open source software system developed by Dimagi to support mobile form data entry. CommCare primarily aids community health workers that are working in communities to facilitate information gathering and medical reporting. CommCareHQ provides its own repository to store data and features reporting for the forms that have been submitted. The application has support for J2ME and Android phones as well as tablets. CommCareHQ also provides programmatic APIs to access data and data feeds that actively notify external systems such as MOTECH of incoming information [16]. As part of the work of this thesis we developed a generalized CommCare interoperability module for the MOTECH platform to communicate and exchange data with CommCareHQ.

CommCareHQ is a form based data collection server that provides users with the ability to design forms through a graphical user interface, deploy forms to mobile phones and tablets and generate reports based on form submissions. CommCareHQ's foremost feature is the capability to deploy XForms that are based on the JavaRosa specification to mobile devices using Android as the preferred operating system. A mobile worker logs in to CommCare on their phone or tablet to fill out forms that are submitted to the server or saved locally on the phone for later submission. CommCareHQ acts as a data repository storing longitudinal “case” data from these forms that is persisted and altered through the natural course

of a use case. CommCare is not the only mobile form data collection tool; other open source projects such as Open Data Kit (ODK) allow Xform submission without a similar longitudinal case model [34].

The work flow for a typical CommCare use case is as follows: a form is submitted through a mobile phone or tablet, which creates, alters or closes a case, then CommCareHQ notifies external systems of these form submissions as well as any of the changes to the case. For example, a pregnancy case begins with a registration, continues with visits, and concludes with a birth or termination of the pregnancy. In this data model the case is the uniquely identifying feature rather than a particular patient or individual. Patient identifiers can be included in the case or forms as an arbitrarily named data element, for example, health_id was used in the Ethiopia implementation's pregnancy registration forms. Figures 2.5 and 2.6 are screen shots of the Ethiopia project's pregnancy application that were taken in an Android emulator running CommCare 2.0.

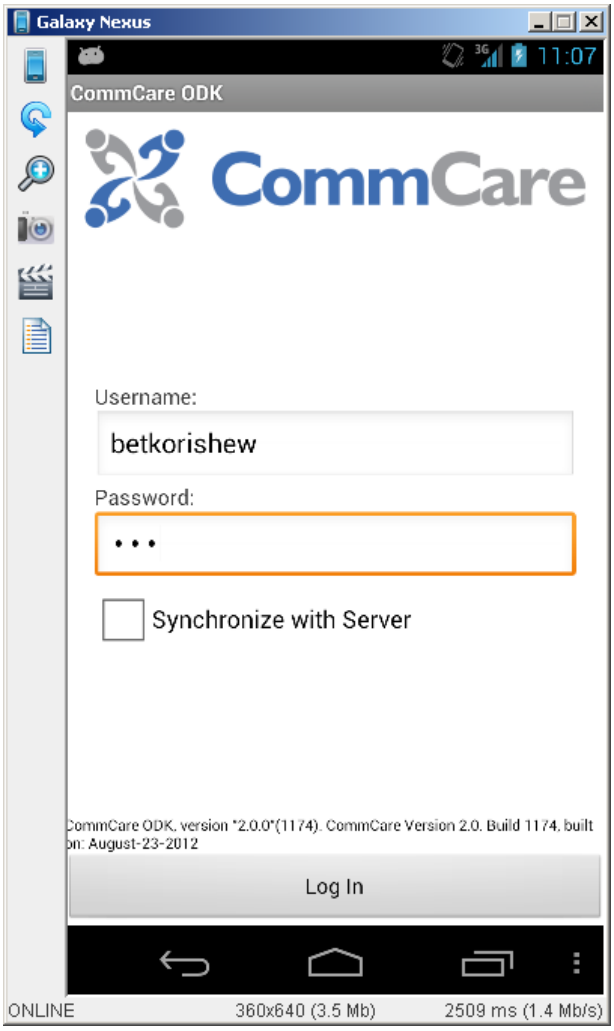


Figure 2.5: CommCare user login screen shot

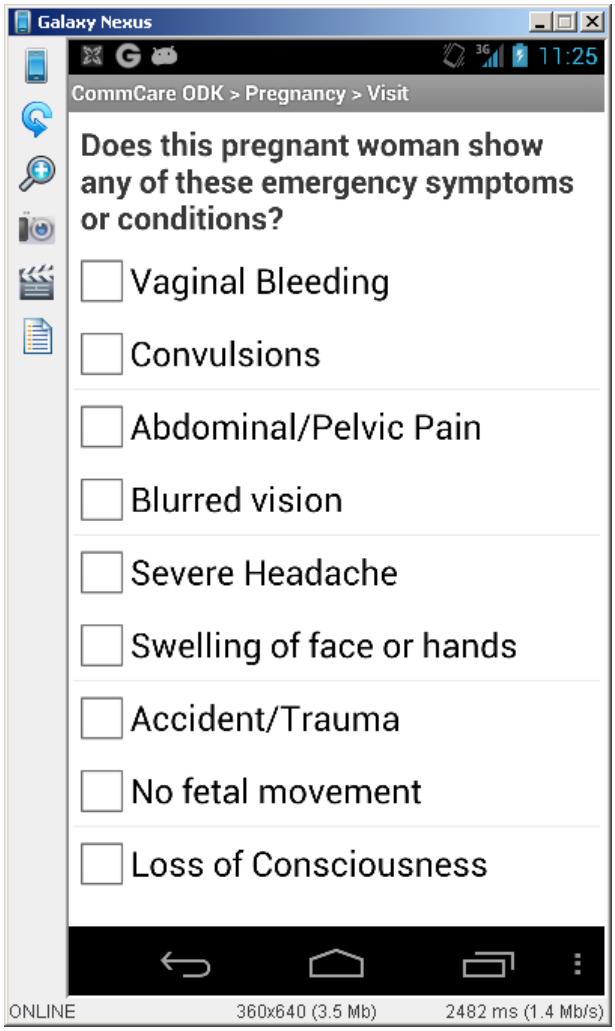


Figure 2.6: Pregnancy form emergency signs list screen shot

2.3.2 - CommCare's Case data model

CommCareHQ's data model is based on the notion of a “case” that lasts for the duration of the logical use case, such as for the duration of a pregnancy. CommCareHQ's case model is an XML specification, illustrated in Figure 2.7.

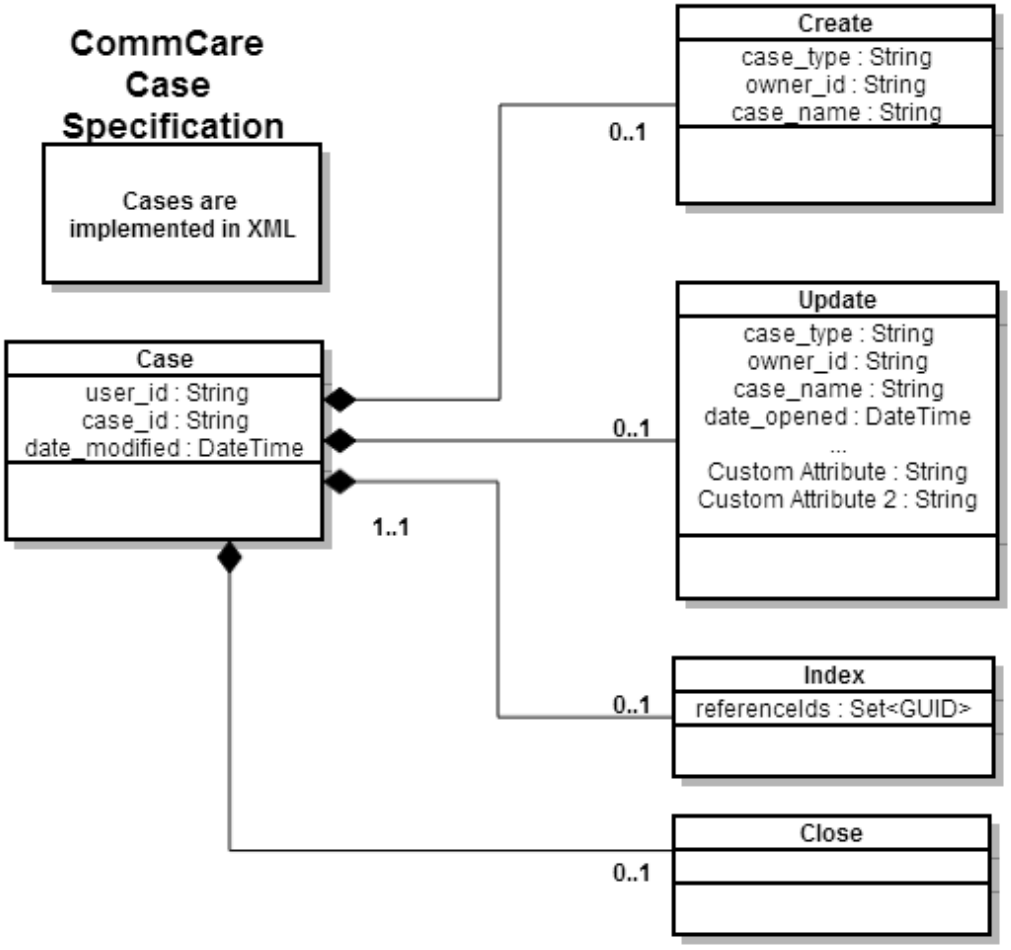


Figure 2.7: CommCare Case specification

The case block encapsulates several other data elements and

operations that are applied to the case, such as creating a case or updating a case. A uniquely identifying case id is part of every case. All cases must begin at some point with a create element. The update element allows a form to update the case information such as case type, add arbitrary case attributes such as “EDD”, or update a preexisting case attribute. The close element contains no data and instructs the case to be set to closed. Index elements are for opening referrals or sub cases related to the case.

The case model distinguishes CommCare's data model from other simple survey tools because its fields are persisted across multiple form submissions. Each form acts as an entry point of data into the case and all of the fields persist throughout the case's duration. The fields may be updated by form submissions and the new value will continue through the case's life cycle. These case variables can be used to make decisions for future questions in the same form or future forms.

2.4 - OpenMRS

OpenMRS is an open-source medical record system primarily used in developing countries to target health care needs and delivery for medical practitioners. OpenMRS's data model supports many of the common health care notions: patients, medical providers, medical encounters such as a visit or surgery, along with medical observations that record medical or other data about a patient. OpenMRS also provides the ability to define a medical concept dictionary to support observation

data. Figure 2.8 is a screen shot of one of the concepts in OpenMRS used in the pilot project. During the work of this thesis, we utilized OpenMRS as a back-end data store for our medical information collected from CommCareHQ forms [17].

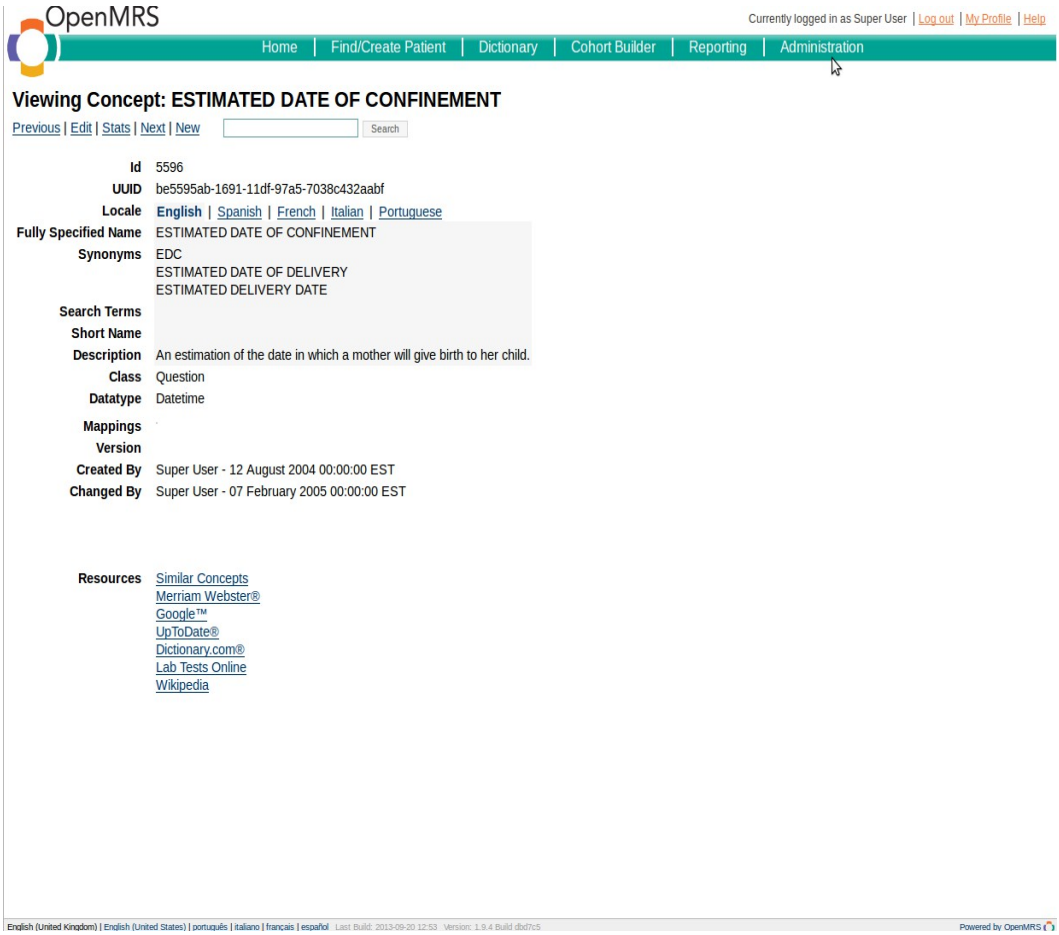


Figure 2.9: Example OpenMRS medical concept from a medical dictionary

2.4.1 - EAV

OpenMRS uses an entity-attribute-value (EAV) data model for many of its entities. The EAV model describes attributes of entities that have potentially many possible values but typically will only have a subset of these values [18]. An example of the EAV data model in OpenMRS is

the patient entity, which developers and administrators can extend by adding arbitrary attributes and values. Another example is the concept dictionary which can grow arbitrarily large, but for any given observation there is only one associated concept. For a full view of the OpenMRS data model, see the link from reference 19.

2.4.2 - Providers

Providers represent individuals involved in the health care delivery process. OpenMRS models providers as person entities that are given the role of provider within the OpenMRS system. Each medical encounter within OpenMRS requires that a provider was involved in the process. Providers as an entity play a key role in the Ethiopia implementation and as part of the work of this thesis we addressed the management of provider data and identity between CommCareHQ, OpenMRS and MOTECH.

2.4.3 - Locations

Locations in OpenMRS are physical places where patients may be visited by a health provider. A client of OpenMRS might also store geographical regions as a location, such as the state of Maine or the city of Portland. Locations may also be hierarchical: Portland resides within the county of Cumberland which is contained by the state of Maine. Locations play a key role in our mapping of data between CommCareHQ and OpenMRS.

2.5 - Interoperability between software systems

An increasing number of software systems need to support methods to communicate information with other external systems. Interoperability has been described as: “The capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units” [20]. Interoperability is an important research area of computer science and it is estimated that inadequate interoperability within U.S. Facilities costs $15.8 billion per year. Lack of interoperability often leads to the duplication or re-development of equivalent features and can lead to monopolies or market failure. There is a greater push for interoperability in the Open-source community by promoting and adopting well understood and open standards. From a health systems perspective, there is a growing desire for interoperability between facilities and even between health devices used by patients. However, competition often has encouraged inter-compatibility within a vendor's own products and the biomedical industry is still in the early stages of fostering and developing true interoperability between health technologies [20].

Interoperability comes in different forms, including syntactic interoperability and semantic interoperability. Syntactic interoperability is the ability of two or more systems to communicate and exchange data. Standards including XML, SQL and JSON help facilitate this process.

Syntactic interoperability is a necessary precursor to achieving semantic interoperability. Semantic interoperability is the ability of two or more systems to exchange and interpret meaningful information that can be used to generate useful results. In order to achieve semantic interoperability, both systems often have to share a common information model or standard that allows for mapping between concepts. A common example of semantic interoperability in health systems is when two systems are able to communicate and map between health concepts. As part of the work of this thesis, we explored syntactic interoperability between MOTECH and other systems. Semantic interoperability was also investigated but currently none of the external systems communicating with MOTECH exchange well defined semantic information [20].

Interoperability between systems can be implemented in a number of different ways, including across various protocols such as HTTP or FTP. Communication between two systems can be as simple as transferring a file across FTP or it may be facilitated through a complex system of web services protected with sophisticated security and code contracts. There is a large body of research on the integration of software, including the integration of enterprise systems. This body of research helped inform the design of our architecture when interoperating within our own enterprise (MOTECH) and with other systems such as CommCareHQ and OpenMRS.

2.6 - Mismatch between data models

Software systems often experience incompatible data models when interacting with other systems or technologies. One of the more well known areas of research in data integration is the object-relational impedance mismatch that occurs between object-oriented programming languages and relational databases. Another major area of research is in enterprise information integration, where there is a unified view of the data across a software enterprise [21]. During the course of this thesis, we investigated enterprise information integration between the domain models of CommCareHQ, MOTECH and OpenMRS [22]. These domains each had their own validation constraints, data formats and identity for data entities.

Enterprise information integration attempts to solve many problems related to integrating disparate systems, such as different persistence strategies, different formats or interpretation of the data and different identifying characteristics of entities. One common approach to integrating these external systems is to utilize a practice known as data transformation. In data transformation, data from an incoming source is converted to a different format for the target system. The crucial step for the data transformation is the data mapping stage, where an algorithm is used to map input data elements in a consistent way to the output source [23]. Technologies such as XLST follow this pattern of data transformation, whereby an XML document is converted into a different XML document or other data type such as an HTML file.

Data integration across systems in an enterprise architecture allows systems to stay uncoupled from one another while communicating and exchanging data. When systems such as CommCareHQ and OpenMRS do not directly communicate with one another, mediators such as MOTECH can communicate with each individually and act as a messaging mediator that helps wire the systems together in a unified way.

2.7 - OpenHIE

Other health organizations have been attempting to solve many of the problems facing MOTECH. Our project's solutions were informed by these efforts and the MOTECH platform may make use of technologies developed by these groups. MOTECH has already leveraged other systems such as OpenMRS and CommCareHQ to offload much of the burden of development and responsibility to groups that specialize and have expertise in various health information domains.

One major effort in the open-source health information domain is the Open Health Information Exchange (OpenHIE) project. The OpenHIE project aims to provide a sophisticated medical record system that includes repositories for a variety of health care entities, including facilities, patients, providers, medical terminology and health records. The OpenHIE project began to support the Rwanda Health Enterprise Architecture (RHEA) project, an electronic medical record system for primary and secondary care. The exchange provides a re-usable framework for other projects to utilize to meet their health information

needs. The architecture consists of an enterprise master patient index (EMPI) or client registry, a health provider registry, a health facility registry, a health terminology service, a shared health record, a health interoperability layer and any point of service applications that interact with the OpenHIE framework [24].

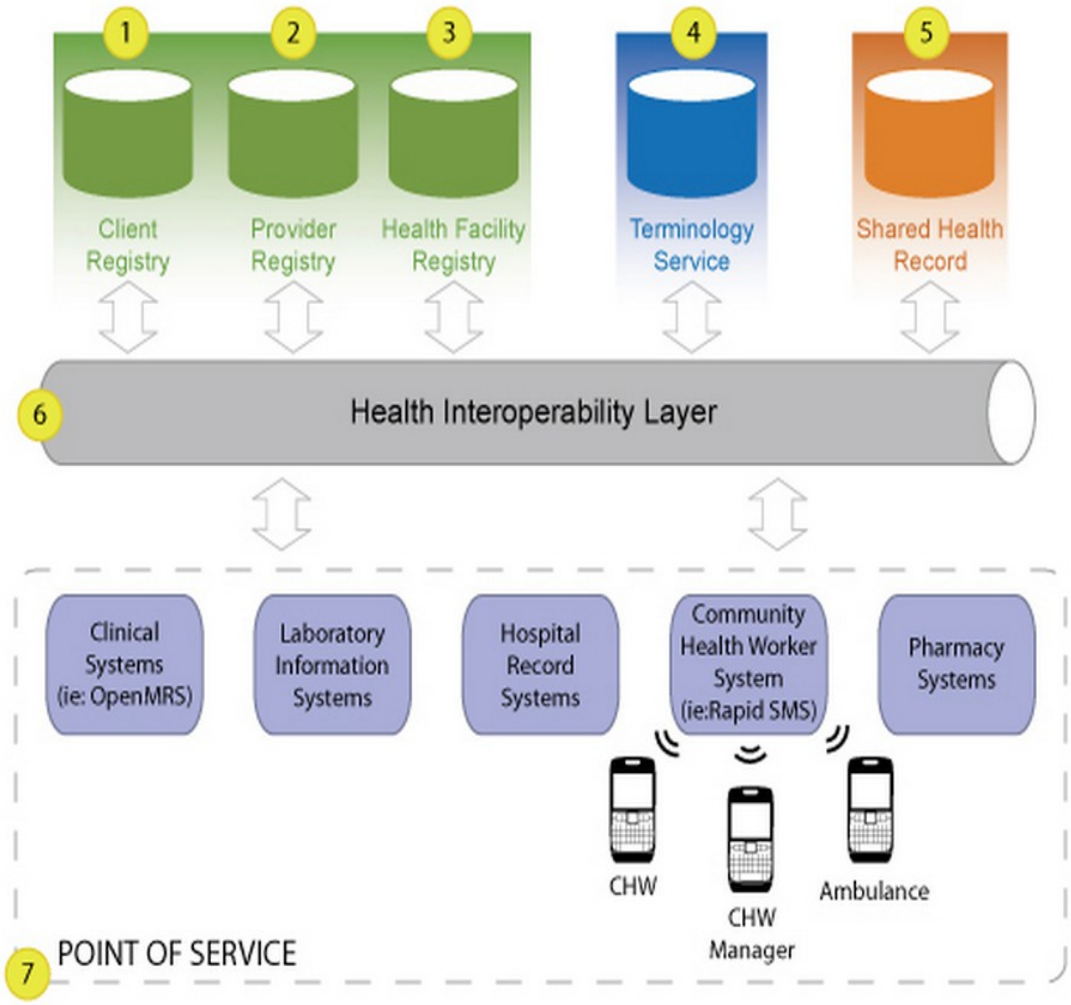


Figure 2.9: OpenHIE Architecture [25]

Each component of the OpenHIE architecture is driven by its own development process that together form a community of projects spanning multiple organizations. For example, the master patient index component

is currently implemented by the OpenEMPI project, which has already been successfully deployed in Rwanda. The OpenEMPI project can perform its functions as a patient index as a stand alone application but will fit within the OpenHIE ecosystem as one potential piece of functionality found in the health IT domain [26]. Another project that implements a component of the OpenHIE architecture is the iHRIS Suite that tracks and manages health providers. The iHRIS project is further broken down into more focused applications, such as iHRIS Manage, which supports Ministry of Health and service delivery organizations in their tracking and management of health workers. Other applications include iHRIS Qualify, iHRIS Plan, iHRIS Retain and iHRIS Train, each adding prospective functionality to a project, such as tracking health worker training, planning and cost retention, predictive modeling tools and database management of registered and licensed health professionals [27].

The Rwanda Health Enterprise Architecture (RHEA) project is an implementation of the OpenHIE architecture and serves as a real world pilot to evaluate the success of a country level electronic health information system using a health information exchange. The project focuses on maternal health care delivery in Rwanda and provides support for maternal health as well as helps identify and define appropriate standards, functional requirements and interoperability needs across several business and foundational domains. RHEA contains all of the layers that are specified in the OpenHIE architecture by using a shared

health record, client registry, provider registry, health facility registry, terminology service, health information mediator and point of service applications such as OpenMRS and RapidSMS [28].

2.7.1 - Potential OpenHIE-MOTECH Interoperability

The MOTECH project is primarily focused on providing support for health IT related use cases in developing countries. Many of the requirements MOTECH faces overlap with the functionality the OpenHIE project plans to provide. Interoperability between MOTECH and OpenHIE could reduce the development work for MOTECH and leverage other open source projects that are focused on providing in depth support and expertise for specific health IT features (patient index, terminology service, facility registry, etc). The MOTECH road map includes eventual interoperability support for extending the system from below so that implementers can define their data models as well as interoperate with the external data models of other systems such as OpenHIE.

The OpenHIE effort helped inform some of the design in our provider and location repository modules described in later chapters. As part of this thesis, we explored the development of basic interoperability repositories within MOTECH that would maintain, track and convert entities between different open source systems such as CommCareHQ and OpenMRS.

III. IMPLEMENTATION AND METHODOLOGY

3.1 - Collaborative Ethiopia Pilot Project

During the work of this thesis we have successfully designed, implemented and deployed a pilot project in Ethiopia in collaboration with Andy Kanter from Columbia University. The project has consisted of several “phases” in which a subset of overall pilot requirements are implemented. We have been part of three phases of the project: 1B, 1C and 1D. Phases 1B and 1C were completed and deployed in Ethiopia and project funding has delayed the deployment of phase 1D. In order to meet project requirements we needed to develop new functionality and extend the MOTECH architecture. During this process we came across fundamental challenges and gaps in the existing MOTECH system. The details of these challenges and our solutions are described in the following sections.

3.1.1 - Phase 1B

The 1B phase of the Ethiopia project focused on indicator reporting for individual facilities spread across four woredas (a district-like geographical location). Health workers assigned to reporting for their facility are supposed to submit one health indicator report per week that consists of various aggregate health data such as number of births,

deaths, and other morbidity indicators. At the end of each week an e-mail is dispatched to regional supervisors that lists the health facilities that did not receive a report that week in addition to the date of the last report from each facility. To support requirements for phase 1B we extended MOTECH with a new module to interoperate with CommCare and an Ethiopia-1B module for implementation specific code.

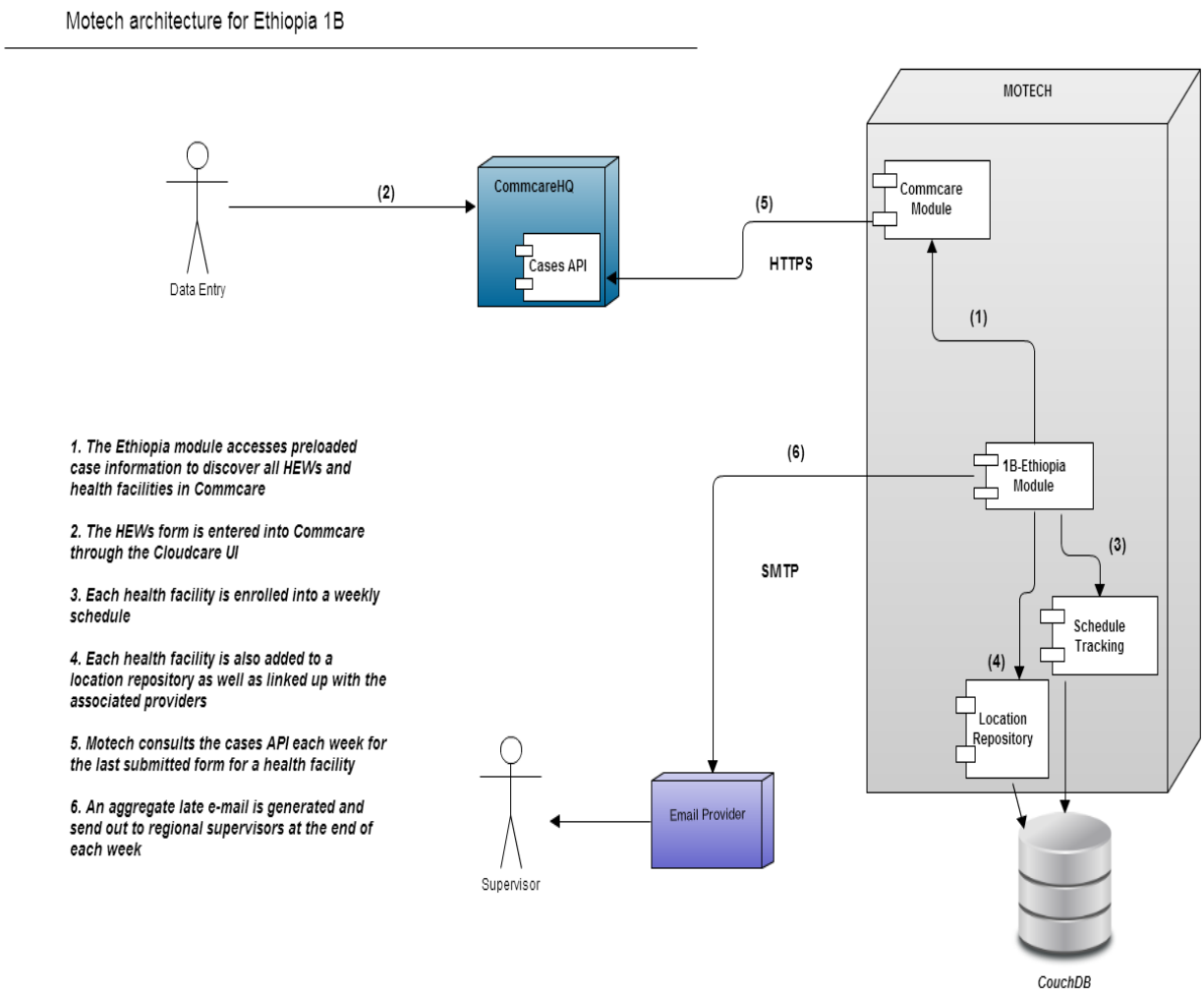


Figure 3.1: Ethiopia phase 1B Architecture

The MOTECH architecture was extended with an additional

Ethiopia-1B module with the primary purpose of maintaining a list of health facilities and alerting supervisors at week's end. Alerts are dispatched for facilities that have not had an assigned health extension worker submit an indicator report for the previous week. The Ethiopia-1B module primarily utilizes services being provided by the CommCare module within the platform. The schedule tracking module was also leveraged with future customizable reporting schedules in mind.

Phase 1B uses e-mail notifications to supervisors to inform them about the adherence of regional health facilities that are scheduled to submit weekly indicator reports. MOTECH's event system was utilized to generate non-compliance events for health facilities when a form had not been submitted for that week. These events were aggregated by region and used to construct each regional e-mail at week's end. E-mail messaging is not a native feature to the MOTECH platform so functionality was added to the 1B module to support this requirement. Since the work performed for the implementation, the MOTECH platform has added basic e-mail functionality.

3.1.2 - Phase 1C

The 1C phase of the project focuses on individual level data collection for pregnant mothers and their children. We were given health forms from a Millenium Villages Project (MVP) in Kenya to use as a proof of concept for this phase [29]. We utilized three different forms that were designed on CommCareHQ: a pregnancy registration form, pregnancy visit form, and child registration form. When each type of form is submitted to CommCareHQ from a mobile device or tablet, MOTECH is notified of the submission and maps the health information into an OpenMRS server. A “nice to have” feature also requested was to push OpenMRS changes back into CommCare's case model. During the work of this thesis we extended MOTECH's architecture with new modules to support a mapping process between CommcareHQ and OpenMRS.

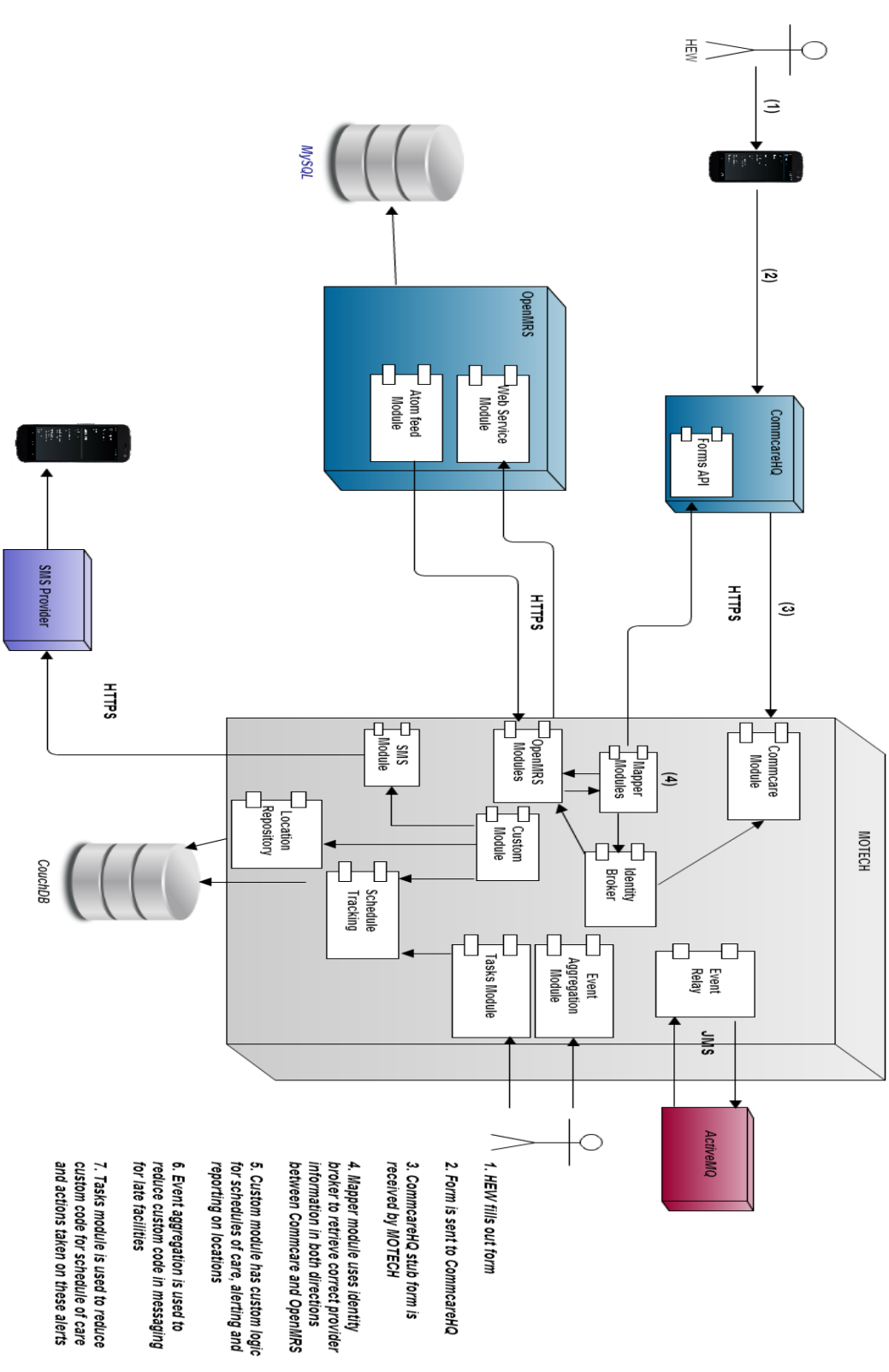


Figure 3.2: Ethiopia phase 1C architecture

In order to shuttle CommCare form data data into OpenMRS, we developed a new mapping module specifically to transform XForm data into MOTECH's medical record terminology entities by utilizing a declarative mapping process. The details of this mapping process are described in section 3.3. We also developed a module to transform OpenMRS medical events into case update actions to update the corresponding case information on CommCareHQ. Instead of using the same module, a separate and independent mapping module was developed to transform medical data, such as an observation update, into an XForm that is programmatically generated and submitted to CommCareHQ automatically.

3.1.3 - Phase 1D

The 1D phase of the Ethiopia implementation expands the requirements of the project to include messaging and alerting based on actionable health events and outcomes. We have identified two approaches for fulfilling these requirements within the MOTECH system: custom code that extends the MOTECH architecture and using the task module's graphical interface to wire together predetermined triggers and actions. The task module is described in section 4.3.2.

Examples of messaging requirements

In Phase 1D of the Ethiopia project, messages are sent out to patients and providers based on actionable medical data. Below are a few

examples of messaging requirements for the project:

1. Weekly ANC Reminder

MOTECH should construct a list of all pregnant women that are assigned to a particular health worker in their 2nd or 3rd trimester of pregnancy who haven't had an ANC visit in the last 5 weeks. This is used to alert the health workers.

2. Birth registration of a child under 28 days

When a new birth form comes in an immediate message is sent to the health worker as a reminder for the child to receive their initial neonatal visit. Six days later, if the child has not received the initial neonatal visit, a follow up message is sent requesting they receive a visit. If a child is HIV exposed, an appointment is created for six weeks after the date of birth. The messages should also be sent to the mother if her mobile number is available.

3. Still birth

If the result of a pregnancy was a still birth the health worker is prompted to follow up on the mother one week later.

During the course of this thesis we successfully developed and deployed the 1B and 1C MOTECH based software. During the implementation of these project phases, three key challenges stood out from the rest: interoperating with external systems, transforming data between domain models and brokering identity between systems. Figure 3.3 gives a full picture of the architecture that we adopted to address these challenges.

Figure 3.3: Interoperability overview

The above is an architectural diagram that illustrates the components needed to achieve interoperability between CommCareHQ,

MOTECH and OpenMRS. The solutions to the three major challenges are all represented in the diagram: the identity broker module used to manage and link the identity of providers between systems, a CommCare module to establish communication between MOTECH and CommCareHQ and a mapper module that transforms incoming CommCare form data into medical record system data for OpenMRS. All of these new components were developed to extend the MOTECH platform. The remainder of this chapter details the approach we adopted for addressing these three challenges.

3.2 - MOTECH's interoperability with other systems

MOTECH interoperates with several other external systems, including OpenMRS, CommCareHQ, Verboice (an IVR provider) and others. During the course of this thesis, we explored and enhanced the interoperability between MOTECH, CommCareHQ and OpenMRS, focusing on access to these systems through their web APIs and mapping data between each. In order to communicate with CommCareHQ, a CommCare interoperability module was developed for the MOTECH platform. Development of the module was informed by several software integration patterns, including Polling Consumer, Publish-Subscribe Channel and Selective Consumer [15].

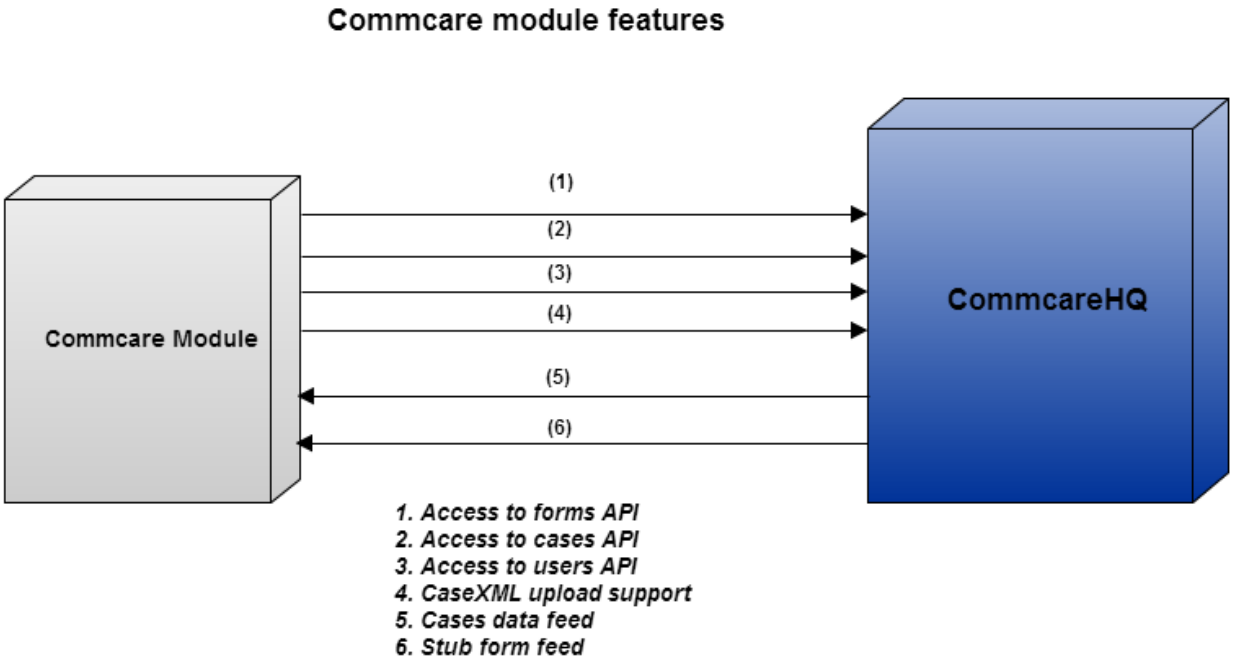


Figure 3.4: CommCare module features

Previous to this work, the MOTECH platform did not include any support for interoperability with CommCareHQ. A MOTECH implementation in Karnataka did interact with CommcareHQ, albeit in an implementation specific fashion through the use of its own custom modules. The code from the Karnataka project was generalized and enhanced to produce a new module that resides in the MOTECH platform that is devoted to interacting with CommCareHQ's APIs as well as its data feeds. Figure 3.4 illustrates the feeds and web APIs the module was developed to initially support.

MOTECH receives real time data notifications from CommCare's

data feeds. This interaction is implemented using the Publish-Subscribe Channel integration pattern. The MOTECH modules that follow this integration pattern are passive and become active upon an external system sending data via an HTTP request. Since these notifications vary from system to system, each module is responsible for handling the incoming request and taking some action as a result. The CommCare module provides an HTTP end point that CommCareHQ sends a request to. The CommCare module raises an event upon receipt of the incoming data, which is then handled by any interested listeners in MOTECH.

MOTECH's event listener system is utilized during the process of active notifications from CommCareHQ. MOTECH's event listeners are implemented using the Selective Consumer integration pattern. The Selective Consumer pattern allows interested pieces of software to specify the messages and data they are interested in receiving; in MOTECH's case these are specific events keyed by their subject. The combination of the Publish-Subscribe Channel and Selective Consumer patterns allows interested MOTECH modules, such as the mapper module, to be aware of and receive specific data from external systems [15]. Figures 3.5 and 3.6 illustrate active notifications made by CommCareHQ and two IVR providers supported by the MOTECH platform.

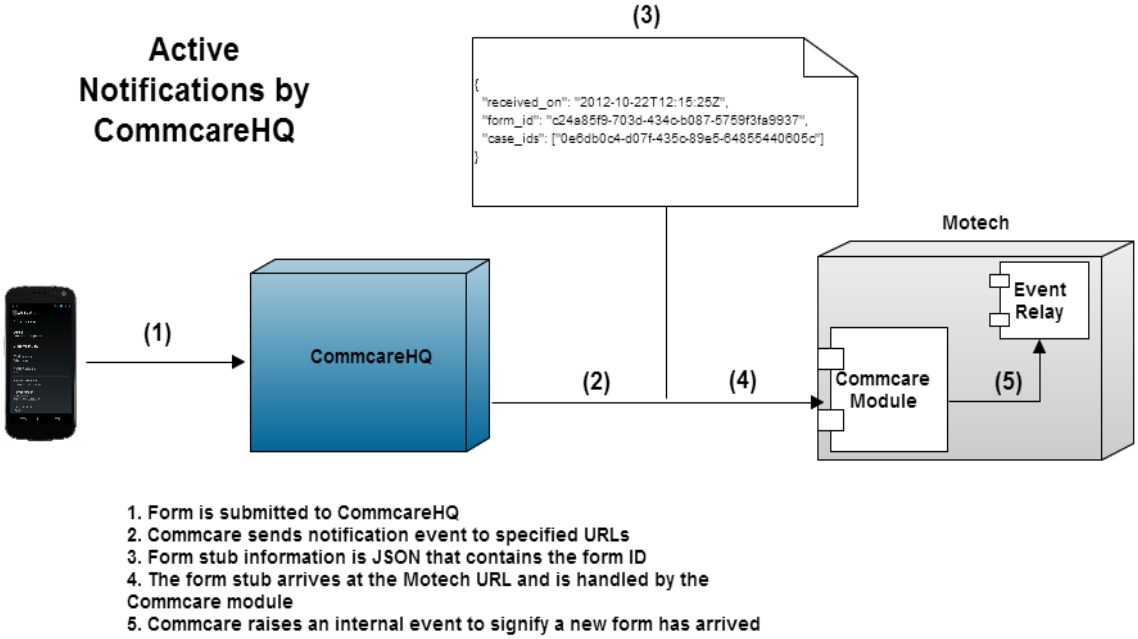


Figure 3.5: Active notifications from CommCareHQ

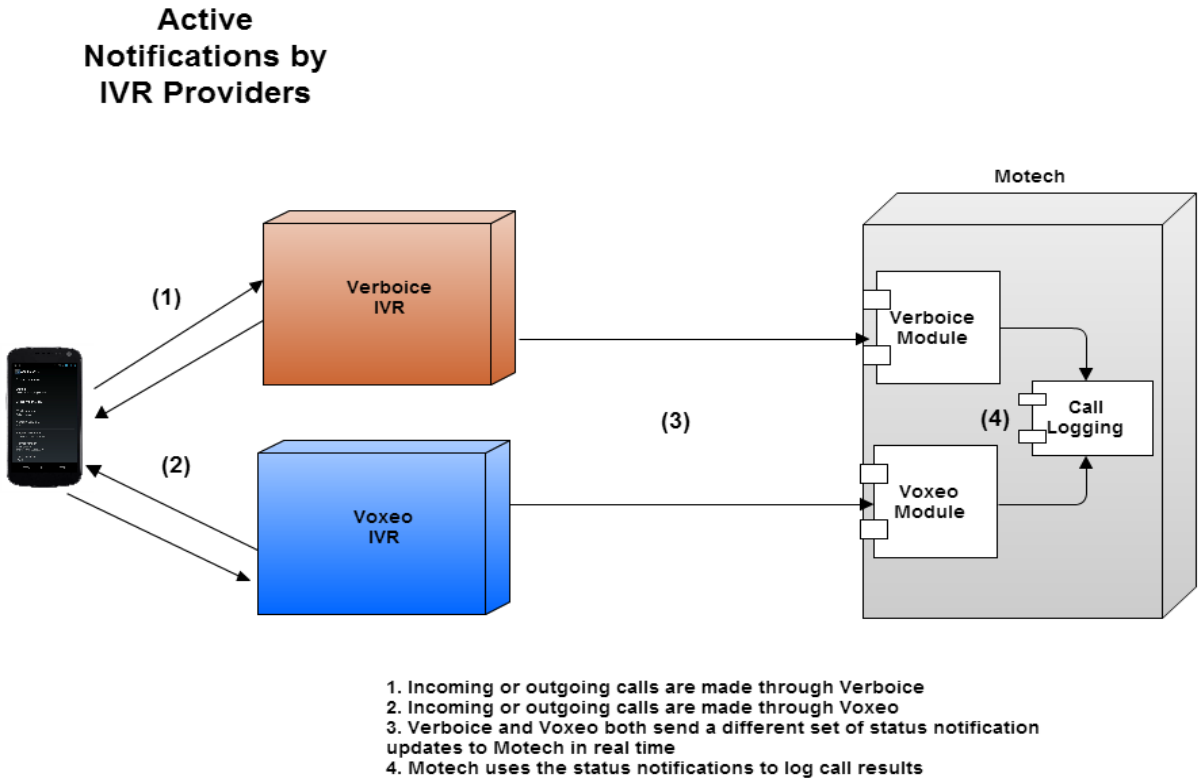


Figure 3.6: Active notifications by IVR providers

When CommCareHQ receives a form that has been submitted by a health extension worker using their mobile phone or tablet, a “stub” containing the id of the form is generated and forwarded to MOTECH (and any other URLs that have been configured to receive the form stub data feed). Once MOTECH receives the stub, an internal event is generated with the accompanying form id information. A mapping module that was developed to extend the MOTECH architecture listens on these events and then retrieves the form data from the forms web API exposed by CommCareHQ. Once the form is retrieved, MOTECH maps this data into medical terminology and uploads the information into an OpenMRS server.

3.3 - Data mismatch and data mapping

Managing the flow of information between CommCare, OpenMRS and MOTECH was a significant challenge in this thesis work. The MOTECH platform has modules to interoperate with both OpenMRS and CommCareHQ. However, OpenMRS and CommCareHQ have no way to directly interoperate and their data models are very different. The CommCare data model is relatively free form and has few constraints while OpenMRS has a relational medical record system with well defined entities and enforced constraints. This data mismatch necessitated that we develop a mapper module for transforming data between medical record data and CommCare's case model. The mapper module is able to

receive CommCare form data from the CommCare interoperability module. Declarative specification provides the rules for transforming form data into medical record terminology. Figure 3.7 outlines the overall mapping process.

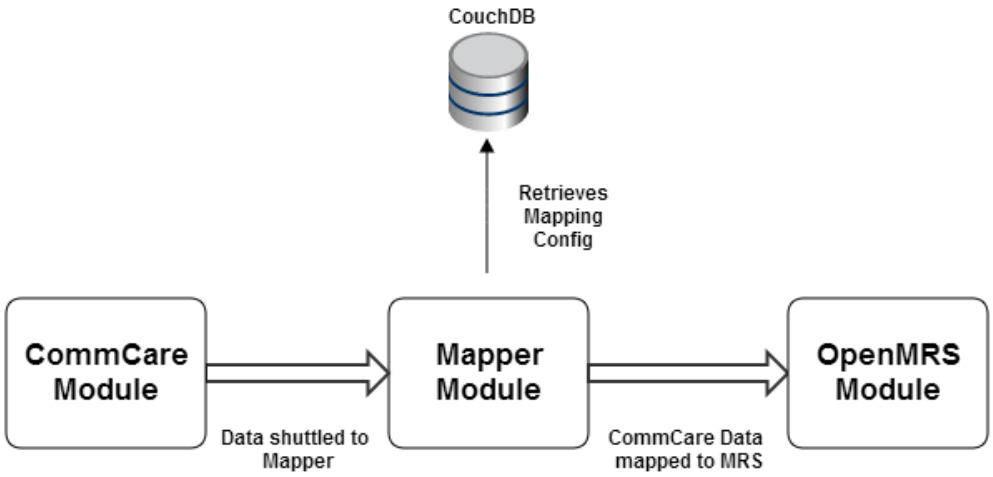


Figure 3.7: Flow of data during mapping

The entry point to the mapping process starts with MOTECH receiving a form stub indicating a new form has arrived and this in turn raises an internal MOTECH event that includes the id of the form. The CommCare to OpenMRS mapping module listens for this event and retrieves the corresponding form from CommCareHQ. MOTECH consults the database for a corresponding mapping configuration for the form to determine if it will be mapped into MRS entities or not. If a mapping configuration is found for the form, the data is converted into medical record data and uploaded to OpenMRS through MOTECH's OpenMRS web services module.

The CommCare module provides the form data to the mapper

module which then retrieves the mapping configuration. The mapping configuration is used to instruct the mapper to convert its data into medical record system entities. The mapper module uses the OpenMRS module to upload these entities into OpenMRS. We made use of the Data Mapper, Mapper and Metadata Mapping software integration patterns to implement the mapper module [32]. The Data Mapper pattern is a layer of mappers that moves data between objects and a database while keeping them independent of each other as well as the mapper. The Mapper pattern is an object that sets up and manages communication between two independent objects while keeping them uncoupled. Metadata Mapping provides the details of mapping data between objects or a database by using metadata configuration. The use of these enterprise design patterns allowed us to design a generalized and extensible mapper module that can be used by a variety of different implementations.

3.3.1 - Mapping configurations and activities

Mapping configurations are broken down into a list of “activities” which represent a particular medical record use case or operation found within the OpenMRS electronic health system. For example, registering a new patient or adding a new encounter are each considered an activity. There are currently two activities supported by the mapping module but it can be extended to provide other activities such as drug orders. Any number of registration and encounter activities, in an arbitrary order, may

be defined in a mapping definition. For example, a mapping configuration could create two patient registrations and two encounters from a single form, which may represent a mother and child patient registration and a medical encounter for each. Figure 3.8 is an illustration of the mapping specification for a pregnancy registration form used during the Ethiopia implementation.

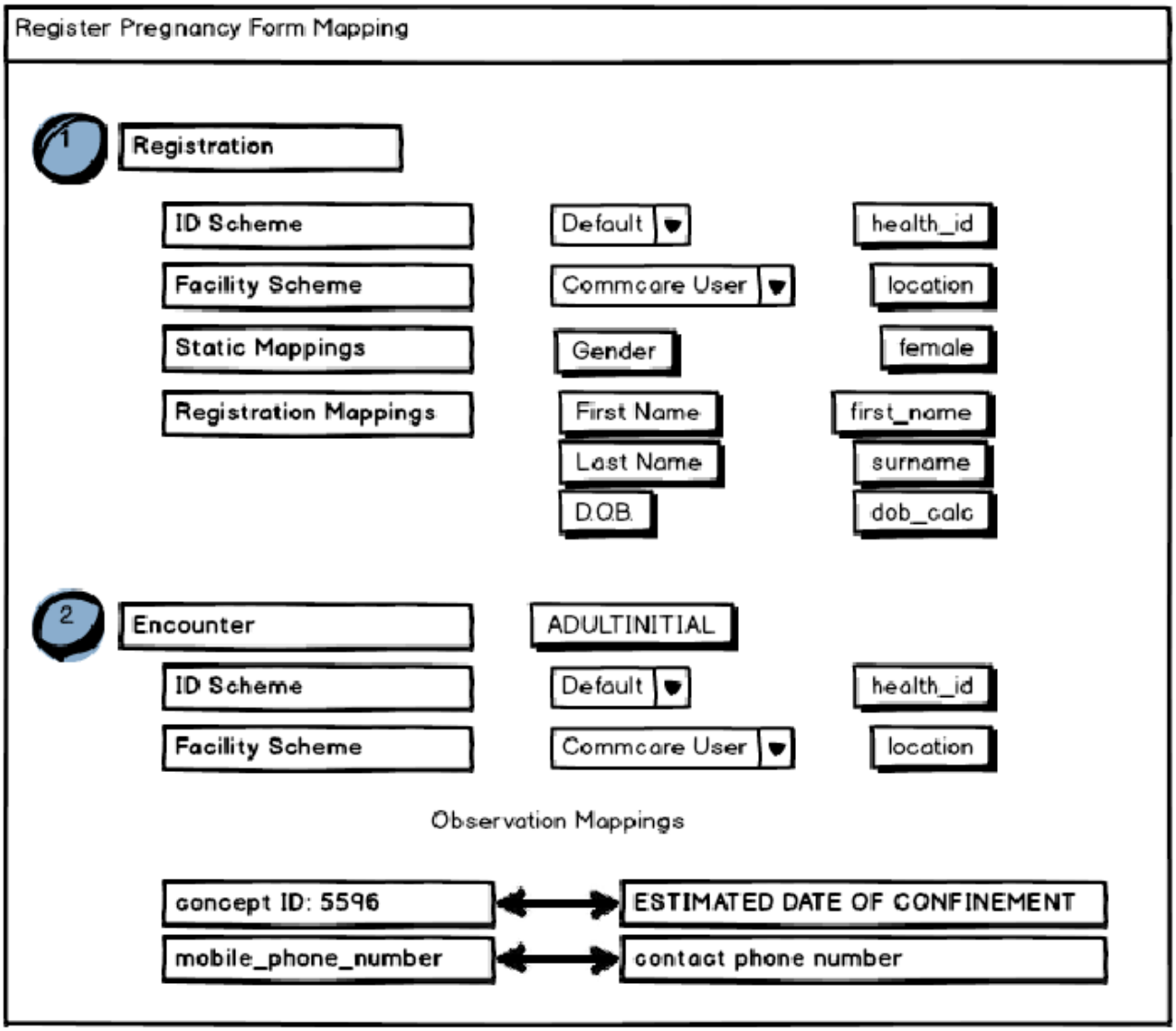


Figure 3.8: Pregnancy registration mapping configuration

The above mapping is divided into two key activities: a registration activity and an encounter activity. Both of these activities are applied against a single form. Each activity is an atomic operation that occurs within OpenMRS. The rest of this section describes each piece of the mapping configuration in Figure 3.8.

Registration activities represent a patient registration or patient information update operation within OpenMRS. A registration activity contains four different fields: an id scheme, a facility scheme, static mappings and mapping values that come from the form instance's patient data. The purpose of an id scheme is to specify the manner in which the MOTECH mapping module discovers the identity of an entity. Currently, id schemes include using a field from the patient or facility ID in the form, a case variable on CommCare, and using a custom attribute that is part of the CommCare web user's data. In the Figure 3.8 the “default” scheme is used with the field name “health_id” to instruct the mapping module that the patient's id should be taken from the field named health_id that is located within the CommCare form instance. An alternative scheme used is the CommCare scheme, which queries for the case information on CommCareHQ and uses a case field, such as patient ID. This scheme was necessary to support the pregnancy visit forms that do not include the patient's health id in the fields of the visit form. Visit forms cannot be selected without prior registering the patient, ensuring that there is always a health_id populated in the case for the visit form mapping process.

The facility scheme likewise is similar in nature to the patient id scheme in that it is used to determine where the facility identifying information should be obtained from. In the above example, the CommCare user scheme is specified, which instructs MOTECH to look up the user that submitted the form on CommCareHQ and check a custom field named “location”. The module uses this location name to map to an OpenMRS facility name.

Static mappings represent data that should be mapped into OpenMRS that is not included in a form or a case field. This mapping functionality was required to support the preexisting Sauri forms we were given to model our phase 1C proof of concept. In the Sauri forms the gender of a pregnant mother is implied, however this information is not part of the form or case and the mapper makes no assumptions regarding the nature of each particular form it receives. This required the mapping specification to include a static mapping of female for its gender, instructing the mapper to use female as the gender for the patient of every pregnancy registration form that arrives.

Registration mappings determine which field names from the form the relevant registration information is retrieved from. In the above example the registration form has a field “dob_calc” and this value is used for the patient's date of birth in OpenMRS. These field values come from each individual form instance and can support any data type that OpenMRS supports.

Encounter activities share some similarities with registration

activities: an encounter activity also has a patient id scheme and facility scheme to determine the OpenMRS patient and facility to use during the mapping process. These activities also include an encounter type to specify the type of encounter in OpenMRS. The main body of these mappings is found in the observation mappings list, for example, the visit form uses ten different observation mappings.

The mapping module provides a way to map incoming form data into medical observations that are linked to a well defined concept dictionary. Observations within an encounter activity are mapped one by one and can vary by their data type as well as the strategy used to link the concept to the OpenMRS concept dictionary. A typical observation mapping requires the observation name and the element name that the value will be obtained from. For example, in the registration form the element edd_calc is mapped to an OpenMRS observation of "ESTIMATED DATE OF CONFINEMENT" (an equivalent of EDD). In addition to mapping by element name, observation mappings also support mapping by a concept id. For example, when an observation mapping is specified by concept id rather than field name, the mapper will look at the form for an element that has concept id "5596" as an attribute of an element in the form. This allows the designer of the form to not have to worry about XML element names and provides a standard for mapping concepts into OpenMRS. Below is an example XML element utilizing the concept_id attribute that is taken from payload of an individual form submission:

```
<edd concept_id="5596">2013-05-06</edd>
```

The element's name is “edd” however, the mapper does not require any knowledge of the element name and the name can be changed to something else entirely as long as it continues to include a concept id attribute.

In addition to mapping field names, the values of an observation do not always map nicely from CommCare to OpenMRS and observation mappings are sometimes needed to map the values. The observation mapping process has a few strategies for mapping values, including whether the data is found in a list, whether the form's value needs to be transformed into a coded answer for OpenMRS or whether to use default behavior for mapping. When an observation is mapped the default behavior of the mapper will take the literal value of the element unless otherwise specified. For the EDD value mapping, the date value does not need to be mapped because the form already stores a date in a format that OpenMRS accepts. However this is not always the case for many of the form's values. When the values are not compatible with OpenMRS, they must be translated to a coded answer that is applicable to the corresponding concept. Figure 3.9 is an illustration of mapping values in the form to coded answers in OpenMRS:

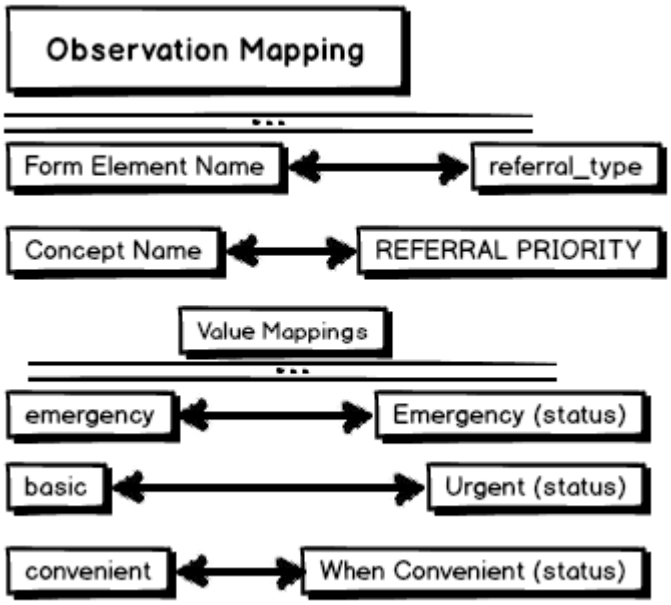


Figure 3.9: Observation value mapping

The CommCare form uses the values “emergency”, “basic” and “convenient” when populating the answer for the form element “referral_type”, but these values are meaningless to the OpenMRS concept dictionary so they must be mapped to a coded answer such as “Emergency (status)”.

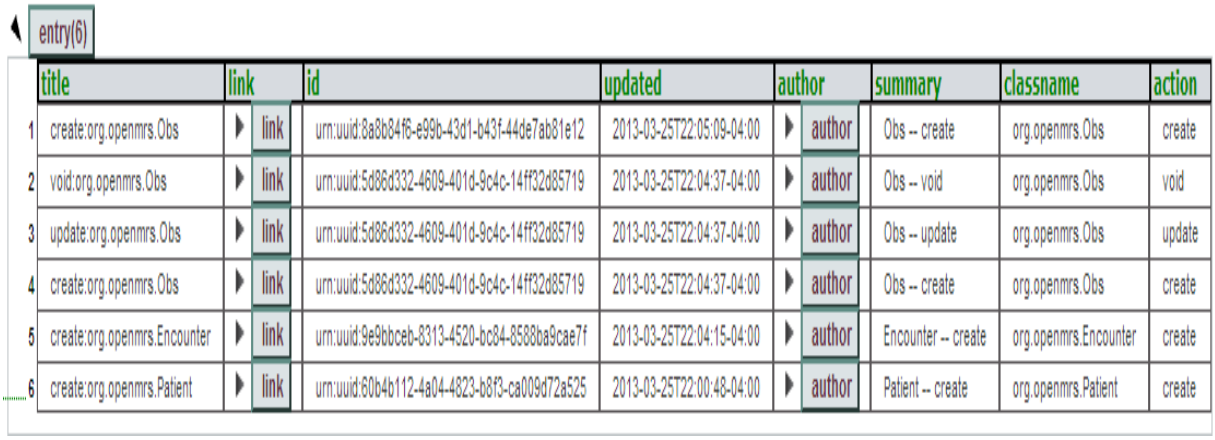
Since we cannot anticipate all future mapping requirements, we have designed the mapping module to support an arbitrary number and combination of activities and made it easily extensible so other activity types can be added when the need arises. Currently registrations and encounters are supported but OpenMRS has other notions of medical actions such as drug orders. We believe the current mapping module could be easily extended to support other activities.

3.3.2 - Mapping from OpenMRS into CommCare

While mapping data from CommCare forms into OpenMRS was a primary project requirement, mapping data in the reverse direction, from OpenMRS back to CommCare was a secondary requirement for when the project moves forward. We utilized a similar mapping process between the two systems with MOTECH acting as a mapping mediator. By polling the atom feed OpenMRS offers we could view changes to entities such as encounters and then construct the necessary case XML needed to update the case within CommCareHQ. Figure 3.10 is a diagram outlining this process.

Figure 3.10: CommCareHQ to OpenMRS mapping architecture

Polling an external system such as OpenMRS is one well known process pattern used for obtaining new data within a modular or federated architecture [15, 31]. Polling consists of sending a request for data at a regular interval determined by the requesting system or subsystem. MOTECH leverages the polling pattern to discover updates to entities within OpenMRS. OpenMRS provides access to these updates through the optional addition of an atom feed module. Once installed, the atom feed module will publish these changes to an interface that can be accessed through HTTP requests.



entry(6)

	title	link	id	updated	author	summary	classname	action
1	create:org.openmrs.Obs	link	urn:uuid:8a8b84f6-e99b-43d1-b43f-44de7ab81e12	2013-03-25T22:05:09-04:00	author	Obs -- create	org.openmrs.Obs	create
2	void:org.openmrs.Obs	link	urn:uuid:5d86d332-4609-401d-9c4c-14ff32d85719	2013-03-25T22:04:37-04:00	author	Obs -- void	org.openmrs.Obs	void
3	update:org.openmrs.Obs	link	urn:uuid:5d86d332-4609-401d-9c4c-14ff32d85719	2013-03-25T22:04:37-04:00	author	Obs -- update	org.openmrs.Obs	update
4	create:org.openmrs.Obs	link	urn:uuid:5d86d332-4609-401d-9c4c-14ff32d85719	2013-03-25T22:04:37-04:00	author	Obs -- create	org.openmrs.Obs	create
5	create:org.openmrs.Encounter	link	urn:uuid:9e9bbceb-8313-4520-bc84-8588ba9cae7f	2013-03-25T22:04:15-04:00	author	Encounter -- create	org.openmrs.Encounter	create
6	create:org.openmrs.Patient	link	urn:uuid:60b4b112-4a04-4823-b8f3-ca009d72a525	2013-03-25T22:00:48-04:00	author	Patient -- create	org.openmrs.Patient	create

Figure 3.11: Example OpenMRS atom feed data

Figure 3.11 shows an example of the OpenMRS atom feed log. MOTECH accesses this log at a configurable interval, such as once per minute, hour or day. The atom feed module within MOTECH will then parse this information and raise corresponding internal events that can be listened on by interested modules. The OpenMRS to CommCare mapping module we developed listens on these events and uploads case

information to CommCareHQ to alter the patient's case data.

3.4 - Identity brokering

The third key challenge identified and addressed as part of the thesis work was the management of entity identity across systems. Provider and location repositories were developed to act as identity brokers between systems. OpenMRS, MOTECH and CommCareHQ each have different identifiers related to providers, locations, and any other entities. The identity broker repository modules allowed for these systems to interoperate on the same entities. The management of health care providers (health systems workers) within the MOTECH suite for the Ethiopia implementation will be used as an example use case throughout this section. Figure 3.12 shows an overview to the identity brokering process.

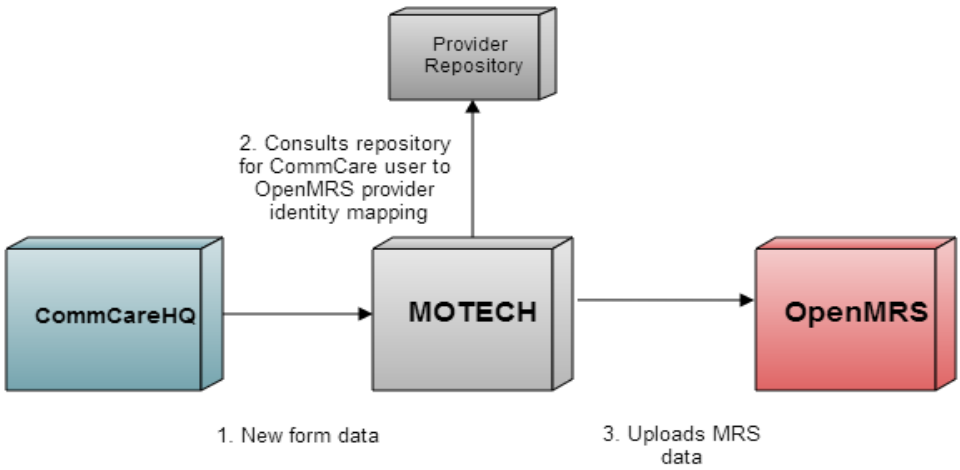


Figure 3.12: Provider identity brokering

Commcare, MOTECH and OpenMRS each have their own notion of a medical provider, its identity and the various data fields associated with that object. In CommCareHQ a provider is an individual who uses the CommCare application for field operations. In OpenMRS providers are represented by person entities that have been assigned the provider role. The provider role represents nurses, doctors and any person involved with delivering health care services to patients. MOTECH has its own lightweight CouchDB implementation of a medical provider. Providers between these systems don't necessarily match up in their data fields or their validation constraints. For example, OpenMRS has particular requirements for a person's demographic information that are enforced, such as name and birth date. In CommCare there are no requirements for a mobile worker so there cannot be a one to one mapping between the two data models without a software piece orchestrating the mapping of data between the two systems. In order to address some of these challenges, a new provider repository module was developed to act as a data mediator between systems.

Identifying information plays a key role during the process of data transfer between systems. OpenMRS, MOTECH and CommCareHQ each have their own id fields for their respective provider entities. When a pregnancy form is filled out and uploaded to CommCareHQ, the provider information is captured through a user ID field found in the form. OpenMRS provider ids are internally generated by OpenMRS and a valid

provider is required for medical encounters in OpenMRS. By linking the CommCare provider id and the OpenMRS provider id, the corresponding provider in OpenMRS can be correctly chosen for the encounter from the form.

The provider repository module extends preexisting infrastructure in MOTECH and acts as a wrapper for MOTECH's provider services. The module persists information related to identifiers for the provider found in external systems such as OpenMRS. This allows MOTECH to create and add to a list of identifiers related to the provider. Once a provider has been linked together through this module, any time a form from CommCare or an update from OpenMRS that refers to a provider that has been registered in the MOTECH provider repository, MOTECH can seamlessly access the corresponding provider in the other external system. A similar approach was adopted for identity brokering with locations and facilities between external systems. Figure 3.13 diagrams the process of identity brokering.

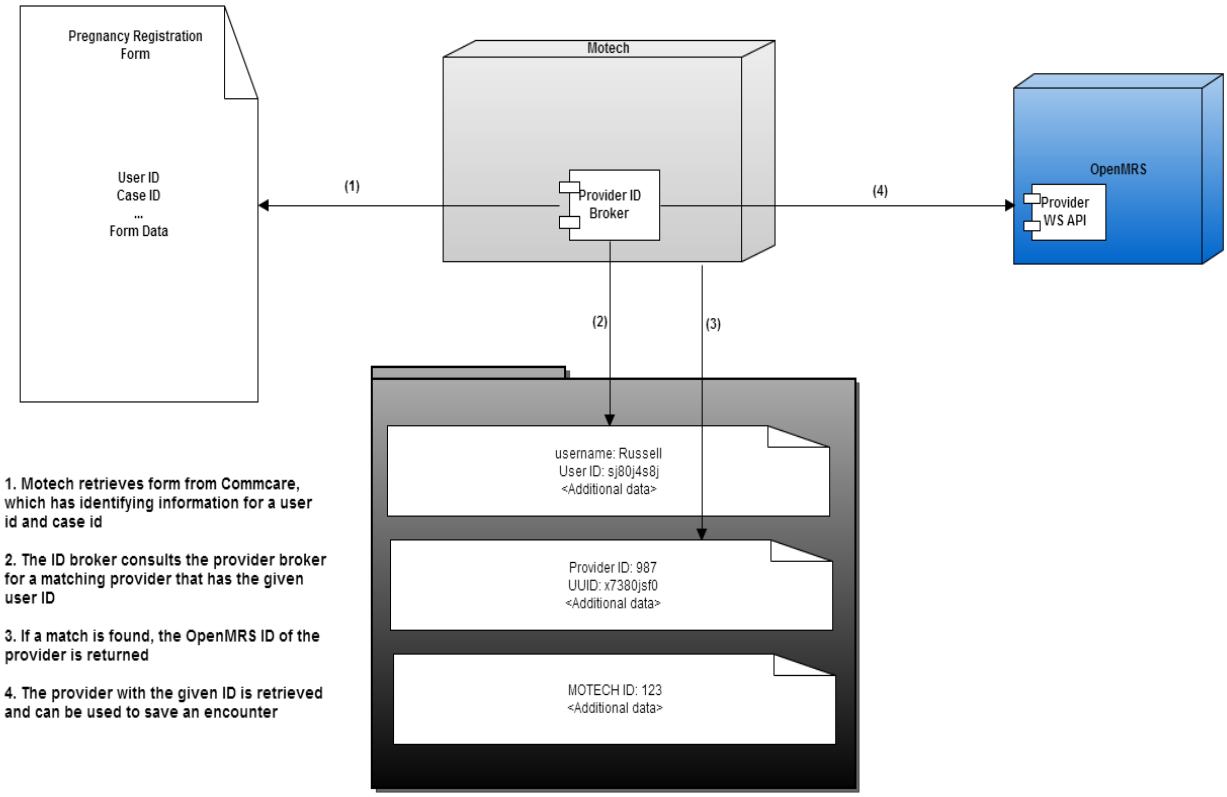


Figure 3.13: Provider identity brokering details

IV. Discussion

Developing and deploying a MOTECH implementation has given us insight into the advantages and some of the key challenges of working with MOTECH as a software platform. In this section, we discuss some of the advantages and challenges of working within a federated platform architecture, the effort required to contribute to platform code from implementation specific work, integration strategies the platform uses and exception handling in a modular environment. We also analyze the software coding burden for implementers and how MOTECH will attempt to further alleviate this burden with future enhancements.

4.1 - Working within a platform architecture

Developing an implementation which uses a software platform such as MOTECH provides many benefits as well as poses a few significant challenges. Platform architectures and frameworks are designed to support many implementations. The goal of most software frameworks is to alleviate developers from the burden of having to implement lower level features themselves [30]. Domain specific frameworks such as MOTECH typically focus on alleviating the coding burden on implementers in one particular domain, such as healthcare or finance. Other domain specific frameworks include the IBM Financial Markets Framework, which provides key capabilities required by almost every financial markets firm, and the Cactus Framework that provides a problem-solving environment designed for scientists and engineers [35,36]. In the case of MOTECH,

features include the implementation of scheduling, medical record, IVR and other systems. By providing these capabilities as a generalized software framework, MOTECH reduces the software coding burden for implementations. The following section discusses the benefits and shortcomings of our use of the MOTECH platform for realizing the Ethiopia 1B and 1C implementations.

The MOTECH platform is a large software system that provides many features for implementers to leverage. The benefits we gained from this project include:

- An event-driven messaging architecture
- An OSGi modular framework that allows adding our own custom modules
- Medical record system functionality
- CommCareHQ interoperability
- Scheduling support
- Web security
- Alerting functionality for future requirements (IVR, SMS)

The Ethiopia pilot project would have had to develop these features from the ground up if the MOTECH platform had not been leveraged. Our project saved significant development effort by using the MOTECH platform and was also able to drive some of the platform's design and evolution by reporting bugs and driving the need for new features within MOTECH.

Working within a framework such as MOTECH can also present challenges for an implementation. Some of these challenges include:

- Sometimes having to maintain a separate platform code base to modify core platform behavior

- Unforseen changes in behavior or the addition of bugs in new releases
- Upfront overhead in learning platform technologies such as OSGi

The two phases of the project we successfully built and deployed required extending the MOTECH architecture to meet functional requirements. The indicator reporting phase required that we develop a CommCare interoperability module and a custom, project specific 1B module for enrolling health extension workers and messaging supervisors. For the individual data collection phase a new mapping module was developed to transform data between CommCareHQ and OpenMRS. Provider and location repository modules were also developed for identity brokering between external systems.

We successfully developed and deployed two phases of a significant research pilot in Ethiopia by leveraging the MOTECH platform. Despite some of its shortcomings, we conclude that the benefits have outweighed the challenges. We were able to successfully utilize and extend the MOTECH architecture for our project. We saved significant development time despite the additional modules and features that were created during the work of this thesis.

The MOTECH platform's alleviation of the coding burden on implementers can be enhanced and informed by implementations that make use of the platform to achieve their project requirements. We used and extended the MOTECH platform to meet real world requirements for the Ethiopia pilot project. During the process of implementing the platform we identified bugs and missing features within the platform that were later

fixed or added to the platform.

An extra level of analysis and design is required to generalize project specific solutions. We generalized and added to implementation specific CommCare code to develop the CommCare interoperability module and add it into the platform. Since the CommCare module was added to the platform, any implementation of MOTECH may now leverage CommCare-MOTECH interoperability. The mapping module we developed during the course of this thesis also saw use and enhancement from another implementation team in Bihar, India that makes use of the MOTECH platform.

Developing generalized solutions for the platform takes more time than the development of a specific solution. The time spent developing generalized solutions was approximately two or three times the effort that would have been needed for a specific solution. However, this extra development effort improves future maintenance of the project and saves development effort for all future implementations. The more general the solution, the more time may be saved, as there are instances where an implementor may have to modify the platform's code and spend development effort on maintaining a separate branch of the main platform. Implementations using a software platform with real world requirements can help discover bugs, missing features, and prioritize development. We believe that implementations of MOTECH should be the drivers behind future development efforts.

4.2 - The design challenges and opportunities for federated systems

Software systems often interoperate with other external systems to save development time and leverage the strengths of another software development's effort. The MOTECH platform can be run as a monolithic, stand-alone application that does not interoperate with other software systems, however, most health information use cases would make use of other software platforms such as CommCareHQ or OpenMRS. The Ethiopia pilot project developed during the course of this thesis utilized both CommCareHQ and OpenMRS. When interoperating with these systems, MOTECH resides within a federated architecture where it depends on outside systems to perform vital operations. In this federated architecture, CommCareHQ provides data collection capabilities, OpenMRS provides a full-fledged medical record system and MOTECH acts as a messaging and scheduling framework that helps broker and manage incoming and outgoing data between the various systems. The MOTECH platform is able to save significant time and development by leveraging the strengths and significant development effort already put into these other software systems. In the following section, we describe some of our integration strategies for achieving a federated architecture.

4.2.1 - Interoperability Integration Strategies

MOTECH interoperates with several external systems, including OpenMRS, CommCare, various IVR and SMS providers, CouchDB, ActiveMQ and others. In order to communicate with each of these

systems, several different integration strategies have been employed: polling an external feed, listening for active notifications, querying a web API, embedding an external library, communicating with a broker through JMS channels, and providing RESTful web services for outside systems to interact with Motech [15]. In the following sections we describe a few of these strategies as well as potential strengths and weaknesses for each that we have identified.

Polling an external feed provides two main advantages: it allows for the specified timing and scheduling of polling to reside within MOTECH and for recovery in the event the data could not be read. MOTECH can choose when to poll for data within OpenMRS, which publishes its events to the feed in real time. This might entail that data is processed only at night or during lower load periods but leaves open the opportunity for real time polling as well. In the event that reading the data fails, MOTECH can also be modified to control its own retry mechanism, unlike active notification systems where the control of exceptional behavior and retries lies with the external system.

Solving the technical issues of establishing communication with polling is not sufficient though and ambiguous or cyclical notifications between external systems still needed to be addressed. The atom feed module in OpenMRS notifies subscribers of the underlying changes to each instance of an object in its database. In practice, an observation may be updated to reflect a revision to its value and the client (MOTECH) only cares that a single, previously existing observation has been updated.

However, in OpenMRS an updated observation consists of a new observation object with the updated value accompanied by an update and void of the previous observation. The atom feed will publish each of these changes as an atomic event. MOTECH then has the burden of trying to understand these changes. Currently MOTECH does not have a way of identifying when an observation is actually new or has instead only been updated, since both actions within OpenMRS will result in a freshly created observation object.

The second complication that arose was the possibility of cyclical updates between CommCare and OpenMRS if two-way mapping is used in MOTECH. The atom feed from OpenMRS has no way of notifying subscribers of where each update to the database originated (such as from the OpenMRS user interface or the web services module). This presents a problem for the two way mapping process because CommCare data is uploaded to OpenMRS through its web services and these OpenMRS changes can be pushed back to CommCare since they are discovered within the atom feed notifications. For an interim solution we make an assumption regarding the OpenMRS user that uploads data from MOTECH's OpenMRS web services module. For example, if an “admin” user or “motech” user is utilized by the web services module, then any database operations originating from that user is assumed to have come from CommCare and will not be mapped back from OpenMRS into CommCare when the atom feed module publishes that event.

An active notification strategy is an alternative to polling and

provides the advantages of real time notifications as well as greater scalability. It also comes with the disadvantage of losing control over the handling of exceptional behavior and retry attempts. Notifications occur as they are happening which removes the responsibility from MOTECH to check at intervals for updates. MOTECH will require less processing and less unnecessary web requests as a result of waiting for active notifications. However, this exchange of responsibility of notification places the handling of delivery failures or exceptional behavior on the external system, which an implementation does not always have access to or control over. For example, if CommCareHQ attempts to deliver a form to MOTECH and the connection fails, CommCareHQ attempts retries with exponential back off and eventually gives up. MOTECH developers have no control over this retry logic or its interval. When MOTECH polls for data, such as a form from CommCareHQ or changes in OpenMRS, the MOTECH implementer has fine grained control over when and how often to retry the data. The number of failed attempts and the time of failure can also be stored on MOTECH's end when polling rather than using an active notification strategy.

One other interoperability challenge faced was maintaining a data entity such as a provider in two disparate repositories by keeping the data synchronized between the two systems. If information regarding a provider changes in OpenMRS or the web user information changes in CommCareHQ, the changes should be synchronized. One method of accomplishing this is choosing a particular repository to act as the master

of the information. The master of the data would always prevail when there are conflicts between systems (such as phone number with the OpenMRS provider and CommCare web user, which is used for alerting the provider). Another alternative is that any change is propagated to the other system immediately, therefore both repositories are acting as peers with the latest change to an entity taking precedence over the older, stale entity in the other system.

A current hurdle for peer to peer maintenance of providers is that CommcareHQ does not send notifications when information has changed for a mobile worker and there is no way to discover when his or her information was last changed. This makes propagating changes as they occur impossible, and a polling mechanism must be used to periodically update the provider in OpenMRS. Figure 4.1 is an example diagram of polling CommCareHQ periodically and updating provider information in OpenMRS.

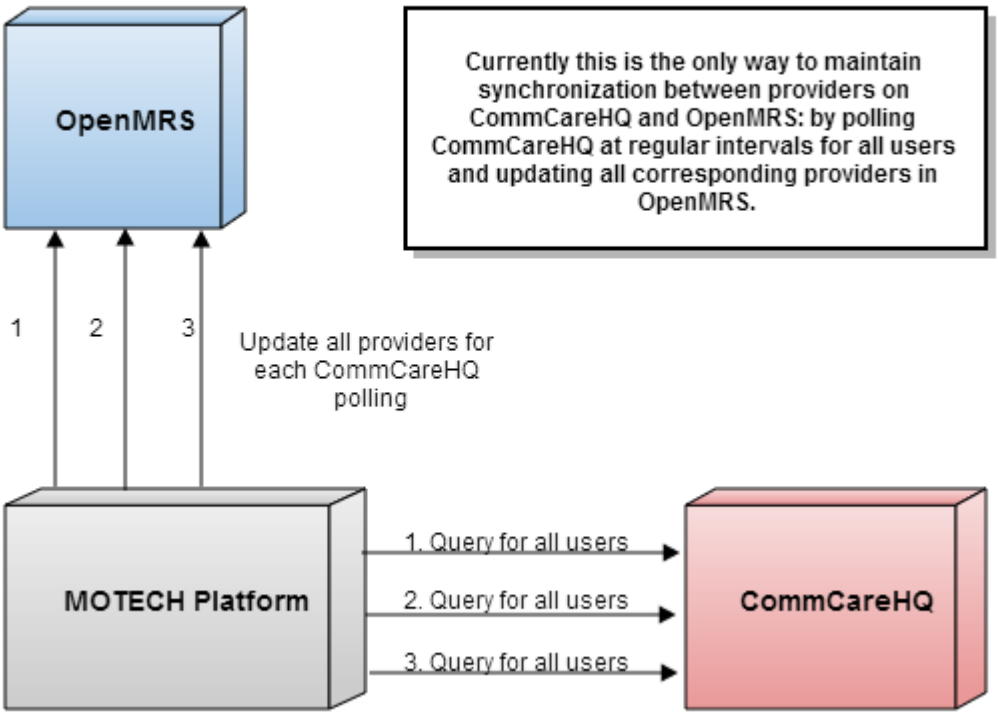


Figure 4.1: Synchronizing providers across systems

Translating information between disparate data models between software systems is another interoperability challenge we encountered. MOTECH's data model is based around traditional medical records with key identifying attributes and first class fields such as name, birth date, gender, etc. CommCare's case XML model offers the advantage of tracking longitudinal data for a single use case over time but is a free form data model that does not align with MOTECH's structured data model. Part of the work during the thesis was aimed at solving this mismatch between CommCare's data model and MOTECH's by developing a mapper module that can translate the information between case XML and

medical record entities.

A major difference with the case XML model compared to MOTECH's is that there isn't a specification for identifying information for patients or locations. This identifying information may be added to the case as an arbitrary attribute but is completely outside the scope of the standard. The consuming client, in this case the MOTECH platform, must manage the patient identity and ensure its uniqueness. If health id “12345” was accidentally entered for two separate pregnancy cases, CommCare would accept this as valid and allow two or more separate cases to be opened with the same health id.

Another challenge faced with bridging these data models is that CommCare's cases do not carry much semantic meaning other than a container for a collection of forms and their attributes. Cases represent encapsulated data for some event over time such as a pregnancy or a vaccination schedule. MOTECH's medical record system can store the time medical encounters and observations occur, however these are not grouped into a “case” that delimits the data from other unrelated medical record data associated with the patient. This difference can cause the updating of data to be somewhat ambiguous because an updated case element may be a revision of a mistake or what could be a completely new medical observation. The best work around is to have an “update” form separate from a “visit” form and to map these forms differently. An update form can be for correcting case data that was incorrectly entered, while a visit form can be considered new medical information.

4.2.2 - Error Handling in a Modular Environment

Software systems must have a way to handle and report exceptional behavior, both from within and in their interoperation with external systems. MOTECH currently employs a few different methods of error handling which include the traditional throwing of exceptions, redelivery of misfired events and the raising of exception events. Currently there is little to no automated handling of or recovery from exceptional behavior. In the following section we describe each strategy of raising error notifications and propose an architecture for the MOTECH platform moving forward.

The majority of MOTECH error handling is accomplished through throwing Java exceptions. Some of the exceptions thrown go unhandled and may even break functionality due to listener redelivery attempting to redeliver an event that caused the exception. This can lead to the same action being taken multiple times, such as several SMS messages sent out from a single event listener that continues to throw exceptions. Examples include throwing an exception when an attempt is made to schedule a job with an execution time in the past or when communication failures occur between HTTP end points.

An alternative method to alerting the system that exceptional behavior has occurred is by raising MOTECH events with corresponding information regarding the error. This information could include the time of

the exception, the offending module, the priority, and any informative data related to the operation that caused the exception. Figure 4.2 illustrates both methods of raising exception information within MOTECH.

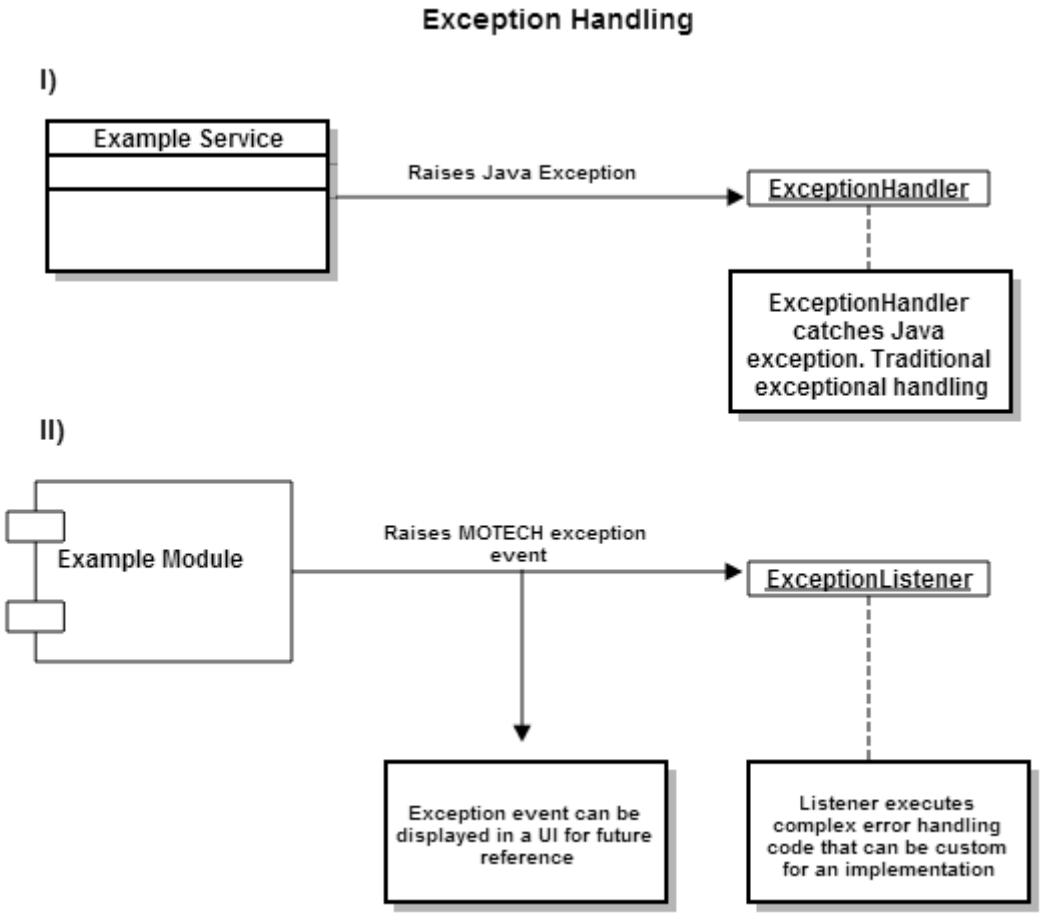


Figure 4.2: Proposed MOTECH exception handling

MOTECH should eventually migrate to a fail-safe system for all of its business logic using internally raised events that do not interrupt the normal functioning of the platform. In addition to providing notification of the error, each event would be given a priority. For example, some exceptional behavior may cripple the system's capability to provide useful functionality on a large scale, while other errors may be the result of a

single patient that has an erroneous phone number. In the case of the former there needs to be immediate notification for any system administrators. Each event should include the module that raised it, along with the time and any pertinent information.

An event driven system for reporting errors allows any custom module to have a hook into handling exceptional behavior in the system, however, some complex exceptional behavior may need human intervention. One example is as follows: a CommCare form arrives that leads to several transactions, such as the creation of a patient and encounter, as well as the enrollment into a schedule of care and a message campaign. If that form is later archived and marked as erroneous, other actions may have already been taken on the patient and simply undoing the original changes may lead to a loss of correct data. We propose that incoming data that leads to several transactions, such as the example given here, has the corresponding transactions packaged together as metadata and persisted for a period of time so that system administrators could view all of the changes and objects related to the form that has been marked as erroneous. This entails that when a user searches for “CommCare form A for patient B”, they might see that the form created an MRS patient, encounter, and enrolled the patient into two different schedules of care. From this interface the administrator could decide what actions are required for restoring data integrity.

4.3 - The software coding burden and the evolution of the platform

The MOTECH platform currently requires significant custom coding to support the requirements of real-world implementations. During the course of the thesis, we had to write approximately 8000 lines of non-platform code (~3500 for the 1B module, ~2500 for the mapper module and ~2000 for the two identity broker modules) to support the current requirements (this excludes the code we developed and submitted to the platform, such as the CommCare interoperability module). If the project moves forward toward implementing the messaging requirements of phase 1D, the coding burden and testing burden would continue to increase.

The goal of a software platform is to lessen the coding burden on other developers by promoting re-use. The MOTECH platform significantly reduced the total development time that would have been required to support an enterprise software system implementation. We estimate that it would take between two and three thousand developer hours of effort for a new implementation that requires Ethiopia 1B and 1C functionality without leveraging the platform. Going forward the platform aims to continue to reduce the coding burden and for the most common use cases, eliminate it entirely by providing UI features, a tasks module and a user-defined data model tool called Seuss. The following sections briefly describe these efforts as MOTECH continues to ease the coding burden on implementations such as the one developed during the course of this

thesis.

4.3.1 - MOTECH UI

Throughout the duration of this thesis a MOTECH user interface has been developed and enhanced. The MOTECH UI plays a crucial role in the future of the project by allowing non-developers to setup, configure and administer the MOTECH server, as well as add in custom functionality for their own implementation. Figures 4.3 and 4.4 are screen shots highlighting some of the UI's current features, including module management and CommCare settings configuration. The long term goal of the project is to provide an interface that allows implementations to be created without the need of advanced developers and custom coding solutions.

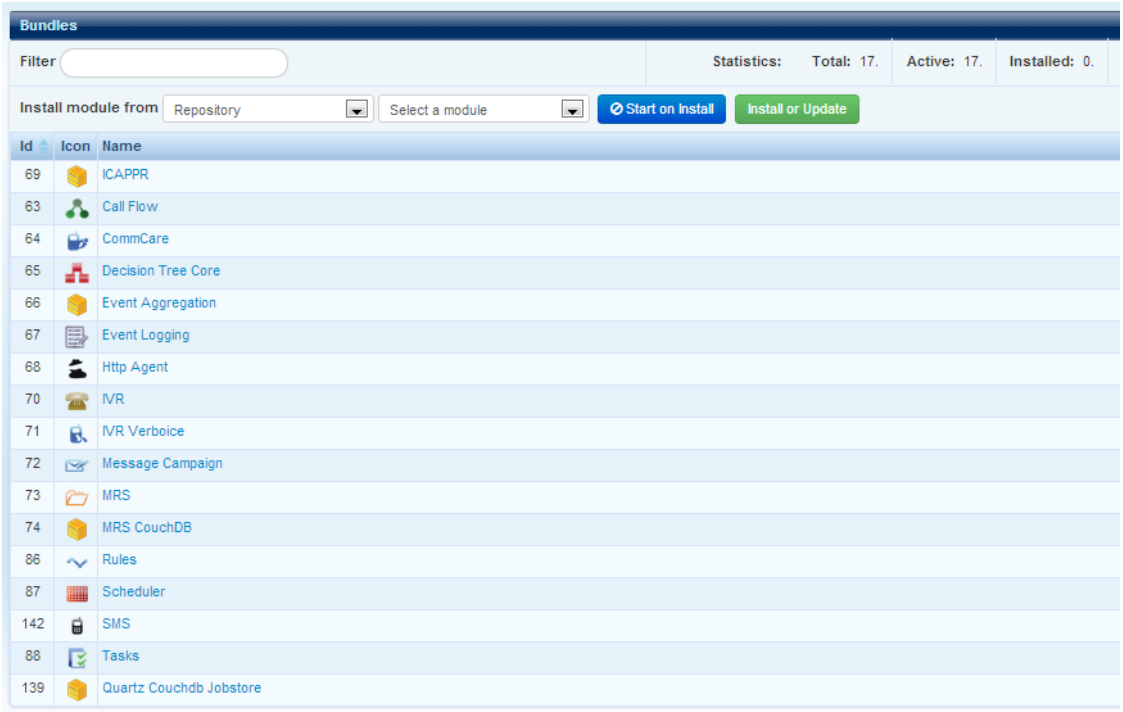


Figure 4.3: MOTECH UI for module management

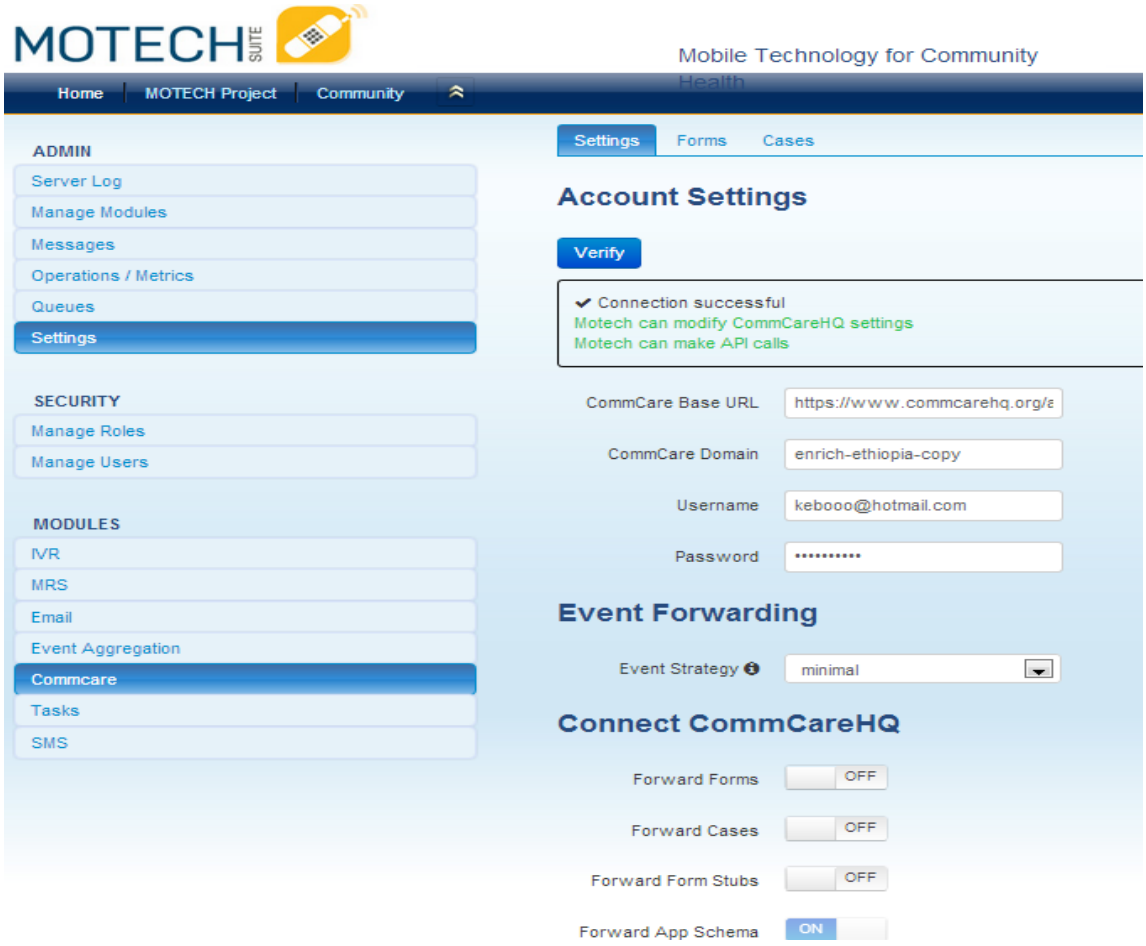


Figure 4.4: MOTECH UI for CommCareHQ settings

4.3.2 - Tasks Module

The MOTECH project is attempting to further alleviate the coding burden on implementers by providing a “tasks” module. The tasks module allows implementers to specify behavior through triggers and actions within a graphical user interface. A MOTECH module registers triggers and actions through the tasks module by providing a declarative JSON specification. Triggers are events within MOTECH with the ability to filter on any of the accompanying payload information, as well as pulling in

extra data from “data providers” such medical record patients. Actions receive the event if the conditions specified have been met and execute some given behavior. An example is when a message campaign event is raised and an SMS message is dispatched based upon the patient's external ID and phone number. Figure 4.5 shows an example of creating a task through the MOTECH UI.

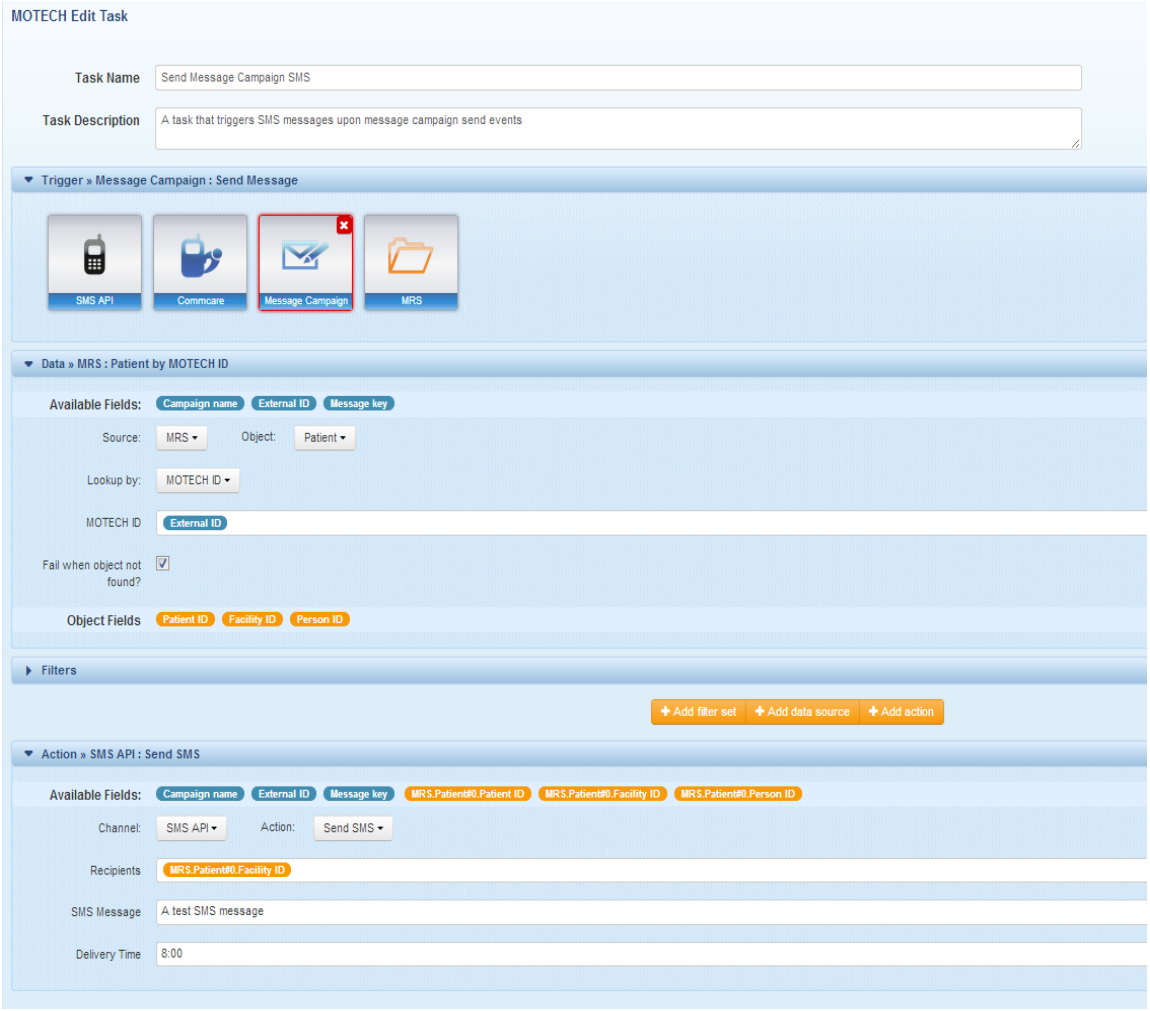


Figure 4.5: Example UI design of a task

Currently the task module only supports very basic features of wiring triggers and actions together. A goal of the MOTECH project is to

allow end users to wire up and execute a number of use cases solely through a graphical interface. The 1D messaging scenarios described by the Ethiopia implementation are currently not supported by the tasks module. Enhancing the module to support the less complex use cases is on the current road map of MOTECH.

4.3.3 - Seuss data model

Implementers that want to extend MOTECH's data model currently must add custom modules with significant portion of the code related to data management. Seuss is a major ongoing development effort of the MOTECH project to create an extensible, user-defined data model that allows for importing and exporting schemas that can be shared across various implementations that fit into the MOTECH system seamlessly. The Seuss data model may feature some of the identifier management between external entities that this thesis tackled for the Ethiopia implementation. Additionally, some of the semantic mismatches in data may be alleviated by allowing users to hook into well known concept dictionaries or other standards based data reference models. The Seuss project is still in the initial design stage and did not impact the work of the thesis, however, it stands as one of the main design changes in the MOTECH platform going into the next year. Combined with the task module, Seuss should further reduce the coding burden on prospective implementations.

V. Conclusion

During the course of this thesis the MOTECH platform architecture was extended to support a health information project in Ethiopia. We successfully delivered and deployed two phases of the project in Ethiopia. In order to meet project requirements we developed key modules that extended and enhanced the MOTECH platform. A CommCare interoperability module was developed for the platform, a custom module was developed to support phase 1B requirements, a CommCareHQ-MOTECH-OpenMRS data mapping module was developed to support 1C requirements, and two identity broker modules were developed to support the project going into the future. We also investigated strategies for systems integration in the platform, the benefits and challenges of working in a federated platform architecture, future improvements for reducing the coding burden for implementations and error handling in a modular environment. The above accomplishments gave us insight into both implementing the MOTECH software platform as well as contributing to its design and architecture for other groups to utilize for their health information projects. MOTECH is an evolving software system which our project and the work completed during this thesis have helped to inform its design and architecture. The MOTECH platform has a bright future and many of the changes contributed from the work during this thesis may see use around the world for years to come.

References

[1] "Health Care System." *Wikipedia*. Wikimedia Foundation, 11 May 2013. Web.

[2] Loechel, Michael. "Electronic Health Records: Implications for IMO State's Healthcare System."*...* State Congress Business Forum, n.d. Web.

[3] "CDA® Release 2." *HL7 Standards Product Brief* -. N.p., n.d. Web.

[4] "How Mobile Phones Are Transforming Healthcare." University of Cambridge, 21 Apr. 2011. Web.

[5] Pigadas, Vassilis, “Enabling Constant Monitoring of Chronic Patient Using Android Smart Phones”, 2011 ACM, May 2011

[6] Smith, Alex D. "Mobile Health Offers Hope to Patients in Africa." *The Guardian*. Bill and Melinda Gates Foundation, 8 June 2011. Web.

[7] DeRenzi, Brian, “Improving Community Health Worker Performance Through Automated SMS”, ICTD '12 Proceedings of the Fifth International Conference on Information and Communication Technologies and Development, ACM 2012

[8] "Cell Phones Can Help Under-Developed Countries Manage Diabetes And Other Diseases." *Medical News Today*. University of Michigan Health System, 17 May 2011. Web.

[9] Hwabamungu, Boroto, “M-Health adoption and sustainability prognosis from a Care givers' and patients' perspective”, SAICSIT '10 Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists, ACM 2010

[10] Sweeney, Chris. "How Text Messages Could Change Global Healthcare." *Popular Mechanics*. N.p., 24 Oct. 2011. Web.

[11] "MOTECH - Mobile Technology for Community Health (www.grameenfoundation.org)." *Scribd*. Grameen Foundation, Mar. 2011. Web.

[12] Bruce MacLeod, Professor of Computer Science at the University of Southern Maine and MOTECH architect

[13] "The OSGi Architecture." *OSGi Alliance*. N.p., 2013. Web.

[14] Fowler, Martin. "Inversion of Control Containers and the Dependency Injection Pattern." N.p., 23 Jan. 2004. Web.

[15] Hohpe, Gregor, and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston: Addison-Wesley, 2004. Print.

[16] "CommCareHQ's Documentation." *CommCareHQ*. Dimagi, 2013. Web.

[17] OpenMRS Community Wiki. 2013. Web.

[18] "Entity-attribute-value Model." *Wikipedia*. Wikimedia Foundation, 17 Oct. 2013. Web.

[19] Direct link from text to domain model: https://wiki.openmrs.org/download/attachments/589829/openmrs_data_model_1.9.0.png

[20] "Interoperability." *Wikipedia*. Wikimedia Foundation, 11 Apr. 2013. Web.

[21] "Enterprise Information Integration." *Wikipedia*. Wikimedia Foundation, 24 Oct. 2013. Web.

[22] "Data Mapping." *Wikipedia*. Wikimedia Foundation, 30 Sept. 2013. Web.

[23] "Data Transformation." *Wikipedia*. Wikimedia Foundation, 27 May 2013. Web.

[24] *OpenHIE Website*. N.p., 2013. Web.

[25] "OpenHIE Architecture." *OpenHIE*. N.p., n.d. Web.

[26] "Open EMPI (Master Person/Patient Index)." *SourceForge*. N.p., n.d. Web.

[27] "How IHRIS Benefits Countries." *IHRIS: Free and Open Health Workforce Information Solutions*. N.p., n.d. Web.

[28] Chrichton, Ryan. "RHEA Phase 1." RHEA, n.d. Web.

[29] "Sauri, Kenya." Millenium Villages. N.p. Web.

[30] "Software Framework." *Wikipedia*. Wikimedia Foundation, 11 Apr. 2013. Web.

[31] "Designing Polling of External Systems." *Patterns for Design Processes*. Outsystems, n.d. Web.

[32] Fowler, Martin. *Patterns of Enterprise Application Architecture*. Boston: Addison-Wesley, 2003. Print.

[33] Macleod B, … "The Architecture of a Software System for Supporting Community-based Primary Health Care with Mobile Technology: The Mobile Technology for Community Health (MoTeCH) Initiative in Ghana." Online J Public Health Inform. 17 May 2012.

[34] *Open Data Kit Website*. N.p., 2013. Web.

[35] "New IBM Software Framework Helps Financial Markets Manage Electronic Trading." *IBM News Room*. IBM, 23 June 210. Web.

[36] *Cactus Code Website*. N.p., 2013. Web.